EU FP7

AMARSi

Adaptive Modular Architectures for Rich Motor Skills

ICT-248311

D 4.1

October 2010 (6 months)

Comparative evaluation of approaches in T.4.1-4.3 and working definition of adaptive module

Authors:
Mostafa Ajallooeian, Sébastien Gay, Auke J. Ijspeert,
EPFL, BIOROB, {first.last}@epfl.ch
S.Mohammad Khansari-Zadeh, Seungsu Kim, Aude Billard,
EPFL, LASA, {first.last}@epfl.ch
Elmar Rückert, Gerhard Neumann,
TUG, {first}@igi.tu-graz.ac.at
Tim Waegeman, Francis Wyffels, Benjamin Schrauwen,
UGent, {first.last}@ugent.be
Andre Lemme, R. Felix Reinhart, Matthias Rolf, Jochen J. Steil,
UniBi, {alemme, freinhar, mrolf, jsteil}@cor-lab.uni-bielefeld.de
Juan Pablo Carbajal, Hidenobu Sumioka, Qian Zhao, Naveen Suresh Kuppuswamy,
UZH, {last}@ifi.uzh.ch

Due date of deliverable	1st September 2010
Actual submission date	15th October 2010
Lead Partner	EPFL
Revision	Final
Dissemination level	Public

Abstract

The goal of this deliverable is two-fold: (1) to present and compare different approaches towards learning and encoding movements using dynamical systems that have been developed by the AMARSi partners (in the past during the first 6 months of the project), and (2) to analyze their suitability to be used as adaptive modules, i.e. as building blocks for the complete architecture that will be developed in the project.

The document presents a total of eight approaches, in two groups: modules for discrete movements (i.e. with a clear goal where the movement stops) and for rhythmic movements (i.e. which exhibit periodicity). The basic formulation of each approach is presented together with some illustrative simulation results. Key characteristics such as the type of dynamical behavior, learning algorithm, generalization properties, stability analysis are then discussed for each approach.

We then make a comparative analysis of the different approaches by comparing these characteristics and discussing their suitability for the AMARSi project.

Contents

Chapter 1

Introduction

One of the main goals of the AMARSi project is to develop a novel control and learning architecture based on dynamical systems for providing robots with multiple degrees of freedom the ability to learn and perform rich motor skills. The architecture will be modular and hierarchical based on coupled adaptive modules that will be used as movement primitives, i.e. building blocks for generating complex movements. The architecture should be scalable, robust, tightly coupled to the compliant mechanics through rich sensory-motor loops, and open-ended.

This is an ambitious goal that requires a careful analysis of the features of the individual modules. In particular, we envision that the modules should present the following features:

- Stable encoding of movement patterns (either as joint angle trajectories, end effector trajectories, or as forces/torques patterns) or internal models (e.g. inverse kinematics and inverse dynamics). Ideally the modules should exhibit attractor properties such that small transient perturbations are rapidly forgotten.

- Ability to produce both discrete and rhythmic patterns. Indeed all movements, e.g. for locomotion, reaching, and manipulation, can be decomposed into superimpositions and sequences of discrete (i.e. with a clear end) and rhythmic movements.

- Possibility to modulate movements: similarly to biological movement primitives (see D.1.1) the modules should be constructed such that a few control inputs can modulate complex output patterns.

- Tight coupling to the mechanics. Since the modules will be used to control compliant quadruped and humanoid robots, they should harness and try to take advantage of the complex dynamics (such as resonant dynamics, for instance) that will result from the interaction between the body and the environment.

- Ability to learn. A module should be capable to learn new patterns, at least before being used (offline learning), but possibly also during use, for online learning.

- Suitability to be used in a hierarchical architecture. We envision that low-level modules will have direct access to actuators, with fast feedback loops and fast time scales, while higher-level modules provide inputs to low-level modules, without direct access to actuators, and with slower time scales. The adaptive modules should therefore be able to work at different time scales and to be coupled to other modules without damaging the stability properties of the complete architecture.

As presented in the Description of Work (DoW), the control architecture of AMARSi will be based on a combination of adaptive modules implemented as dynamical systems (e.g. sets of differential equations). Indeed, dynamical systems offer an ideal representation for motor control and learning, and this for three reasons: (1) they are well-suited for a tight coupling with a mechanical body (e.g. a nonlinear oscillator can be designed to drive, and be entrained by, a mechanical pendulum), (2) they can offer robustness against perturbations (e.g. a dynamical system that exhibits single-point attraction or limit cycle behavior will rapidly forget transient perturbations), and (3) they are well-suited for learning (e.g. non-linear dynamical systems play a key role as filter and fading memory in the reservoir computing paradigm). The consortium brings together an extensive expertise on dynamical systems. Indeed, the consortium has worked with low-dimensional systems (coupled oscillators) to high-dimensional systems (recurrent neural networks), with different regimes (single point attractors, limit cycle behavior, and chaos), and on different applications (control of robots, learning, and computation).

The goal of this deliverable is therefore two-fold: (1) to present and compare different approaches towards learning and encoding movements using dynamical systems that have been developed by the AMARSi partners (in the past and during the first 6 months of the project), and (2) to analyze their suitability to be used as adaptive modules, i.e. as building blocks for the complete architecture that will be developed in the project.

The document presents a total of eight approaches, in two groups: modules for discrete movements (i.e. with a clear goal where the movement stops) and for rhythmic movements (i.e. which exhibit periodicity). The systems for discrete movements are the following:

- Dynamical Movement Primitives (EPFL, Chapter 2)
- Stable Estimator of Dynamical Systems (EPFL, Chapter 3)
- Neural Dynamical Motion Primitives Generator (UGent, Chapter 4)
- Neural Motion Primitive Control (UGent, Chapter 5)
- Neural Dynamic Movement Primitives (UniBi, Chapter 6)
- Planned Motion Primitives using Approximate Inference(TUG,Chapter 7)

And those for rhythmic movements are:

- Dynamical Movement Primitives (EPFL, Chapter 2)
- Adaptive Frequency Oscillators (EPFL & UZH, Chapter 8)
- Neural Central Pattern Generator (UGent, Chapter 9)

The basic formulation of each approach is presented together with some illustrative simulation results. Key characteristics such as the type of dynamical behavior, learning algorithm, generalization properties, and stability analysis are then discussed for each approach. The document is organized such that each chapter has exactly the same structure with same section titles such as to facilitate comparisons. We conclude the document with a comparative analysis of the different approaches (Chapter 10) in which we compare these characteristics and discuss their suitability for the AMARSi project. The main conclusion that comes out of this detailed analysis is that currently there is not a single approach that clearly outperforms the other approaches and exhibits all the desired features needed for the architecture. We therefore propose for the rest of the project to investigate two main directions: (1) the creation of a new type of module that combines interesting features of the different approaches developed so far, and (2) the development an architecture that is hybrid i.e. that combines different types of modules for different functionalities.

Chapter 2

Dynamical Movement Primitives

Mostafa Ajallooeian, Auke Jan Ijspeert
EPFL, BIOROB

2.1 Short Introduction

Dynamical Movement Primitives (DMPs)[1] propose a generic modeling approach to generate multi-dimensional systems of nonlinear differential equations to capture an observed behavior in an attractor landscape. The essence of the presented methodology is to transform well understood simple attractor systems with the help of a learnable forcing function term into a desired nonlinear system. Both point attractor and limit cycle attractors of almost arbitrary complexity can be achieved. Multiple degrees-of-freedom can be coordinated with arbitrary phase relationships. Stability of the model equations can be guaranteed. This approach also provides a metric to compare different dynamical systems in a scale invariant and temporally invariant way.

This model is designed to achieve the goals listed below:

1. Both learnable point attractor and limit cycle attractors need to be represented. This is useful to encode both discrete and rhythmic trajectories.
2. The model should be an autonomous system, i.e., without explicit time dependence.
3. The model needs to be able to coordinate multi-dimensional dynamical systems in a stable way.
4. Learning the open parameters of the system should be as simple as possible, which essentially opts for a representation which is linear in the open parameters.
5. The system needs to be able to incorporate coupling terms, e.g., as typically used in synchronization studies or phase resetting studies.
6. Scale and temporal invariance would be desirable, e.g., changing the amplitude or frequency of a periodic system should not affect a change in geometry of the attractor landscape.

[1] There are different implementation of DMPs. The case discussed here is the system introduced in [1]

2.2 Model Description

The basic idea of this model is to use an analytically well understood dynamical system with good stability properties, and then modulate it by nonlinear terms to achieve a desired attractor behavior. A simple model to use is a damped spring model:

$$\begin{aligned} \tau \dot{z} &= \alpha_z(\beta_z(g-y)-z)+f \\ \tau \dot{y} &= z \end{aligned} \tag{2.1}$$

where τ, α_z, and β_z are positive time constants. Choosing forcing term $f = 0$ gives a globally stable second-order linear system with $(z, y) = (0, g)$ as a unique point attractor. But phasic of periodic choices of f generate nonlinear point-attractor systems and nonlinear oscillators respectively. Since f transforms the simple dynamics of the unforced system, the dynamical system in the equation 2.1 is called the "transformation system".

Forcing term f can have any arbitrary structure. So, normalized linear combination of basis functions (or any other general function approximator) can be used to model it:

$$f(t) = \frac{\sum_{i=1}^{N} \Psi_i(t) w_i}{\sum_{i=1}^{N} \Psi_i(t)}$$

where Ψ_i are fixed basis functions and w_i are adjustable weights. The time definition in the model can also be replaced by a simple first order system (called "canonical system"):

$$\tau \dot{x} = -\alpha_x x \tag{2.2}$$

where α_x is a time constant and x monotonically converges to zero. So, the forcing term becomes:

$$f(x) = \frac{\sum_{i=1}^{N} \Psi_i(x) w_i}{\sum_{i=1}^{N} \Psi_i(x)} \; x \; (g - y(t)|_{t=0}) \tag{2.3}$$

with N exponential basis functions $\Psi_i(x)$:

$$\Psi_i(x) = \exp\left(-\frac{1}{2\sigma_i^2}(x-c_i)^2\right) \tag{2.4}$$

where σ_i and c_i are constants that determine, respectively the width and centers of the basis functions. The modulation term $x \; (g - y(t)|_{t=0})$ is used to 1) have useful scaling properties, and 2) and to make the forcing term vanish when g is reached.

By replacing the discrete canonical system with a periodic canonical system, limit cycle oscillators can be modeled in the same way as point attractor systems. A simple choice is a phase oscillator:

$$\tau \dot{\phi} = 1 \tag{2.5}$$

where $\phi \in [0, 2\pi]$ is the phase angle in the polar coordinates, and the amplitude of the oscillation is assumed to be r. With this, the forcing term for a periodic system is:

$$f(\phi, r) = \frac{\sum_{i=1}^{N} \Psi_i w_i}{\sum_{i=1}^{N} \Psi_i} r \tag{2.6}$$

$$\Psi_i = \exp\left(h_i(\cos(\phi - c_i) - 1)\right) \tag{2.7}$$

where g serves as an anchor point and Ψ is a Gaussian-like periodic function.

Another issue is to extend the model for multiple degrees of freedom. This is simply done by assuming a shared canonical system for all degrees of freedom. A number of

phase coupled canonical systems can also be used for this purpose if coupling all dimensions with one canonical system is not desired.

Learning arbitrary trajectories is done by locally weighted regression. If equation 2.1 is rearranged and the data from the learning demonstration $(y_{demo}(t), \dot{y}_{demo}(t), \ddot{y}_{demo}(t))$ is inserted:

$$f_{target} = \tau^2 \ddot{y}_{demo} - \alpha_z(\beta_z(g - y_{demo}) - \tau \dot{y}_{demo}) \tag{2.8}$$

where f_{target} is the proper forcing term to have y_{demo} trajectory as the output. τ is initialized as movement duration (period) and g is set to the desired goal position (oscillations baseline) for a discrete (periodic) system.

Estimating proper values for w_i to model f_{target} is done employing locally weighted regression (LWR). LWR finds for each kernel function Ψ_i in f the corresponding w_i which minimizes the locally weighted quadratic error criterion:

$$J_i = \sum_{t=1}^{P} \Psi_i(t) \Big(f_{target}(t) - w_i \xi(t) \Big)^2 \tag{2.9}$$

where $\xi(t) = x(t)(g - y(t)|_{t=0})$ for the discrete system, and $\xi(t) = r$ for the rhythmic system. This is a weighted linear regression problem which has the the solution

$$w_i = \frac{\mathbf{s}^T \Gamma_i \mathbf{f}_{target}}{\mathbf{s}^T \Gamma_i \mathbf{s}} \tag{2.10}$$

with

$$\mathbf{s} = \begin{pmatrix} \xi(1) \\ \xi(2) \\ ... \\ \xi(P) \end{pmatrix} \quad \Gamma_i = \begin{pmatrix} \Psi_i(1) & & & 0 \\ & \Psi_i(2) & & \\ & & ... & \\ 0 & & & \Psi_i(P) \end{pmatrix} \quad \mathbf{f}_{target} = \begin{pmatrix} f_{target}(1) \\ f_{target}(2) \\ ... \\ f_{target}(P) \end{pmatrix}$$

The above formula is used for batch (offline) training when all of the demonstration data is present. To have an as-data-comes (online) learning procedure, recursive least squares with a forgetting factor of λ to determine the parameters w_i is used:

$$w_i^{t+1} = w_i^t + p_i^{t+1} \; \xi(t) \; e_i(t) \tag{2.11}$$

where

$$p_i^{t+1} = \frac{1}{\lambda} \left(p_i^t - \frac{(p_i^t \xi_i^t)^2}{\frac{\lambda}{\Psi_i} + p_i^t (\xi_i^t)^2} \right) \tag{2.12}$$

$$e_i(t) = f_{target}(t) - w_i^t \xi(t) \tag{2.13}$$

There is also a number of variations to the introduced model. For example, one may want to have a continues acceleration profile when changing the goal state suddenly. This can be formulated as:

$$\tau \dot{g} = \alpha_g(g_0 - g) \tag{2.14}$$

where g_0 is the discontinuous goal change, while g is now a continuous variable. There are more variations described in [1]. Finally, a summary of the introduced model is listed in Table 2.1.

2.3 Simulation

Here, some numerical tests are presented. Figure 2.1 demonstrates an exemplary time evolution of the equations designed to do a reaching movement. Figure 2.2 shows an exemplary time evolution of the rhythmic pattern generator when trained with a

Tab. 2.1: Summary of the equations for our discrete and rhythmic model equations. The high level design parameters of the discrete system are τ, the temporal scaling factor, and g, the goal position. The design parameters of the rhythmic system are g, the baseline of the oscillation, τ, the period divided by 2π, and r, the amplitude of oscillations. The parameters w_i are fitted to a demonstrated trajectory using Locally Weighted Learning. The parameters $\alpha_z, \beta_z, \alpha_x, \alpha_r, \alpha_g, h_i$ and c_i are positive constants.

Discrete	**Rhythmic**
Transformation System	
$\tau\dot{z} = \alpha_z(\beta_z(g-y)-z)+f$ $\tau\dot{y} = z$	$\tau\dot{z} = \alpha_z(\beta_z(g-y)-z)+f$ $\tau\dot{y} = z$
Canonical System	
$\tau\dot{x} = -\alpha_x x$	$\tau\dot{\phi} = 1$
Forcing Term	
$f(x) = \frac{\sum_{i=1}^{N}\Psi_i(x)w_i}{\sum_{i=1}^{N}\Psi_i(x)}\ x\ (g-y_0)$ $\Psi_i = \exp\left(-h_i(x-c_i)^2\right)$	$f(\phi,r) = \frac{\sum_{i=1}^{N}\Psi_i w_i}{\sum_{i=1}^{N}\Psi_i} r$ $\Psi_i = \exp\left(h_i(\cos(\phi-c_i)-1)\right)$
Optional Terms	
$\tau\dot{g} = \alpha_g(g_0-g)$	$\tau\dot{r} = \alpha_r(r_0-r)$
Default Values	
$c_i \in [0,1]$ $h_i =$ equal spacing in $exp(-\alpha_x t)$ $\alpha_z = 25$ $\beta_z = \alpha_z/4$ $\alpha_x = \alpha_z/3$ $\alpha_g = \alpha_z/2$	$c_i \in [0,2\pi]$ $h_i =$ equal spacing in $\phi(t)$ $\alpha_z = 25$ $\beta_z = \alpha_z/4$ $\alpha_r = \alpha_z/2$ $\alpha_g = \alpha_z/2$

superposition of several sine signals of different frequencies. It should be noted how quickly the pattern generator converges to the desired trajectory after starting out of zero initial conditions.

Figure 2.3 illustrates the spatial (Figure 2.3a) and temporal (Figure 2.3b) invariance using the example from Figure 2.1. One property that should be noted is the mirror symmetric trajectory in Figure 2.3a when the goal is at a negative distance relative to the start state. We will discuss the issue again later in the chapter.

Figure 2.4 depicts a modulation test on a periodic system. As illustrated, once a rhythmic movement has been learned, it can be modulated in several ways (amplitude, frequency, anchor).

2.4 Functional Analysis

2.4.1 Dynamics and Nonlinearity

The introduced dynamics of this system is nonlinear convergent and the nonlinearity is parameterized. The whole dynamics of the system is a result of the interaction between canonical systems, transformation systems, the nonlinear forcing term, and the coupled items. For the discrete system, the nonlinearity of the forcing term is transient, i.e. it vanishes when the goal is reached. However, the nonlinearity of the forcing term for the periodic system is not transient.

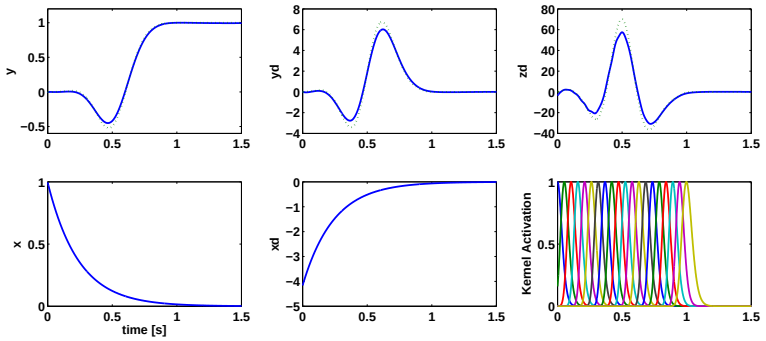


Fig. 2.1: Exemplary time evolution of the discrete dynamical system. The parameters w_i have been adjusted to fit a 5-th order polynomial trajectory between start and goal point, superimposed with a negative exponential bump. The upper plots show the desired position, velocity, and acceleration of this target trajectory with dotted lines, which largely coincide with the realized trajectories of the equations (continuous lines). On the bottom right, the activation of the 20 exponential kernels comprising the forcing term are drawn as a function of time – the kernels have equal spacing in time, which corresponds to an exponential spacing in x.

2.4.2 Attractor

The attractor of this system can be a point attractor or a limit cycle attractor. If the system is designed to have a point attractor, the point attractor is located in $(z, y, x) = (0, g, 0)$. If system is designed to have a limit cycle behavior, the limit cycle is defined by the y_{demo} training input. Both point attractor and limit cycle attractors have global basins of attraction.

2.4.3 Coupling

There are separate transformation systems for each DoF. the subsystems are coupled together to create a multi-dimensional coupled system. The coupling is usually made by having a shared canonical system for all DoFs. Like this, interestingly, the canonical system becomes a central clock, not unlike the assumed role of central pattern generators in biology. It is also possible to have a number of coupled canonical systems, like having two canonical systems, one for the left side and one for the right side of a humanoid robot.

2.4.4 Learning

Type The type of learning is supervised. Policy search has also been reported to learn DMPs with reinforcement learning strategies [2].

Algorithm The used learning algorithm is LWR, but the model is not dependent to the learning algorithm and other general function approximators can also be used.

Mode System accepts both offline (batch) and online (as-data-comes) learning modes provided by LWR.

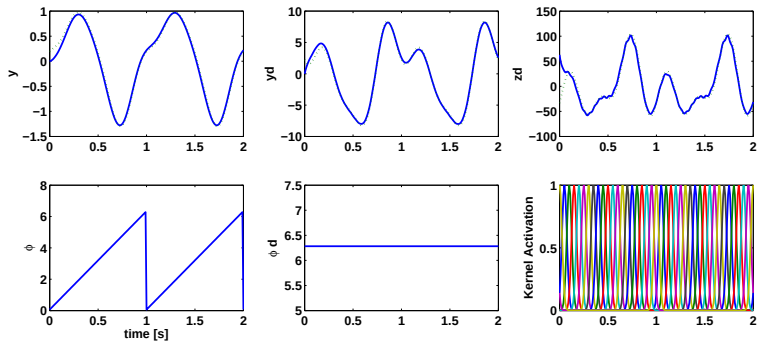


Fig. 2.2: Exemplary time evolution of the rhythmic dynamical system (limit cycle behavior). The parameters w_i have been adjusted to fit a trajectory $y_{demo}(t) = sin(2\pi t) + 0.25cos(4\pi t + 0.77) + 0.1sin(6\pi t + 3.0)$. The upper plots show the desired position, velocity, and acceleration with dotted lines, but these are mostly covered by the time evolutions of $y, \dot{y}$ and $\ddot{y}$. The bottom plots show the phase variable and its derivative and the basis functions of the forcing term over time (20 basis functions per period).

2.4.5 Training Data

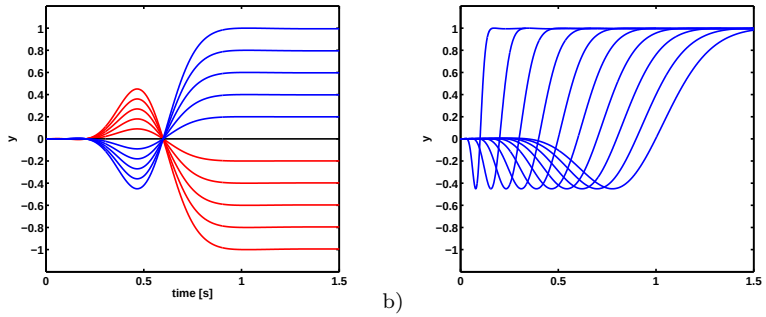
The input training data to the model is trajectories. Generally, the position trajectory of joints with its first and second derivative are presented to the system for learning. It is needed to extract the frequency of the learning data beforehand. If multiple demonstrations of a trajectory exist, even at different scales and timing, they can be averaged together in the locally weighted regression learning after the f_{target} information for every trajectory at every time step has been obtained.

2.4.6 Generalization

Here, the discussion about the generalization issues is separated for discrete and periodic systems:

- **Discrete:** For the discrete system, goal is modulated online by changing g. So, the goal state can even be set to values that are not seen in the training phase. Examples of changing the goal state (even to negative values) are illustrated in 2.3. Generalization about the initial state is also possible. But since the position of the goal state is relative to the initial position, goal position needs to be updated relatively if the initial state is changed. If the generalization is defined as the ability to change the geometry of the trajectory (signal shape) in the recall phase, it can be said that this system needs a re-learning step. Nevertheless, the learning is one-shot and low cost, so re-learning can be done instantly.
- **Periodic:** Defining what is generalization for a periodic system is a bit tricky. If by generalization, modulation of frequency, amplitude, etc is meant, they are addressed in 2.4.7. If the generalization is defined as the ability to change the geometry of the trajectory (signal shape), again, a one-shot re-learning is required.

An important issue about the generalization is if the outcomes for new goals are acceptable or not. For the proposed model, this, to some extent, is dependent on the choice of coordinate systems. See Figure 2.5 for an in-depth example.



a) b)

Fig. 2.3: Illustration of invariance properties in the discrete dynamical systems, using the example from Figure 2.1. a) The goal position is varied from -1 to 1 in 10 steps. b) The time constant τ is changed to generate trajectories from about 0.15 seconds to 1.7 seconds duration.

2.4.7 Modulation

The introduced system is able to smoothly modulate working parameters without affecting the geometry of the basin of attraction. For the discrete system, speed is modulated by changing τ. For the periodic system, the frequency modulation is controlled by τ while the anchor (mid-point of oscillations) is modulated by changing g. Additionally, the amplitude of the generated output can be modulated by changing r in the forcing term.

2.4.8 Sensory Feedback Integration

The transformation system defined in equation 2.1 can accept different coupling terms (additive term). Additionally, couplings can be made to the canonical system to affect the phase state. These coupling terms can be defined as a function of the sensory feedback. Obviously, the definition of the sensory feedbacks are case-based and depends on the application. For example, a drumming task may need temporal coupling based on sensory feedback, while a bipedal walking task totally needs sensory feedbacks including information about balance. See [3, 4] for some examples.

2.4.9 State Variables

There are three x, y, and z state variables in the standard definition. To have smooth modulation in each parameter (e.g. goal), an additional state variable for each is required. y and z state variables are separate for each DoF, while x is generally shared between DoFs. If the model is extended for N-DoFs, and each dimension has k smoothly modulated variables, then the total number of state variables is $(2+k)N+1$.

2.4.10 Robustness and Adaptation to Perturbations

To define the reactions to perturbations, consider the time independent model of the proposed system. The time independent model will be a three dimensional model with x, y and z axes. When talking about time independence, it is meant that the dynamical behavior of the whole system can be depicted in a quiver plot independent of any explicit external time definition.

For the instances of this three dimensional space that are on the desired trajectory, the behavior of the vector field is to evolve x and follow the desired trajectory. When

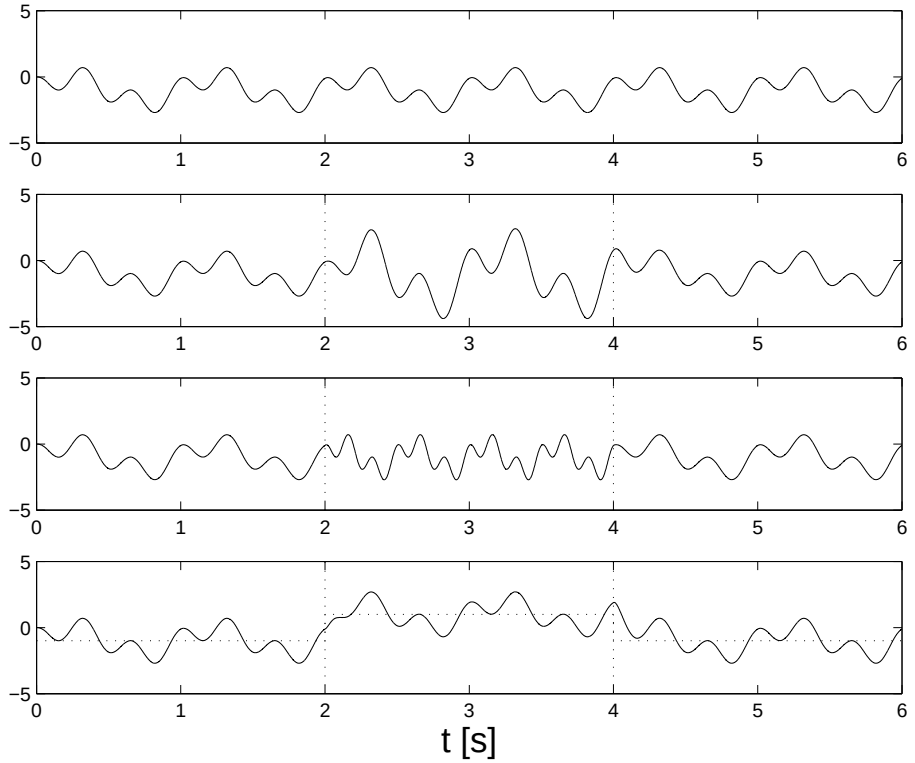


Fig. 2.4: Modulation of a rhythmic dynamical systems: Top: original trajectory, which was generated as an arbitrary example. Second from top: r_0 multiplied by a factor 2.0. Third from top: τ multiplied by a factor 0.5. Bottom: g increased by 2.0 (as indicated by the horizontal dotted lines). All modulations are done between between t=2 and t=4s.

talking about perturbation, it means that the value of the y for the current instance has been changed (for both temporal and spacial perturbations). So the natural behavior of the system is to asymptotically reach the proper (x, y, z) state. This is exactly what the is happening with respect to equation 2.1.

To clarify, here, the reaction to a perturbation is to asymptotically reach the trajectory while the phase variable (x) is evolving. So, if there is a joint lock (temporal perturbation blocking the movement in one DoF), the subgoal (point to reach on trajectory) evolves. As a result, the system bypasses a part of the y trajectory and tries to reach the evolved trajectory state. There are pros and cons regarding this behavior. This behavior is good since it does not force the system to go back to the state before perturbation. So the state variables are always evolving toward reaching the goal. However, this behavior is problematic if skipping a part of the designed trajectory is hazardous.

To counter this behavior, one can incorporate both temporal and spatial coupling, and introduce the coupling terms (with α_e, k_t and k_c as constants)

$$\dot{e} = \alpha_e(y_a - y - e) \tag{2.15}$$

$$C_t = k_t e \tag{2.16}$$

$$\tau = 1 + k_c e^2 \tag{2.17}$$

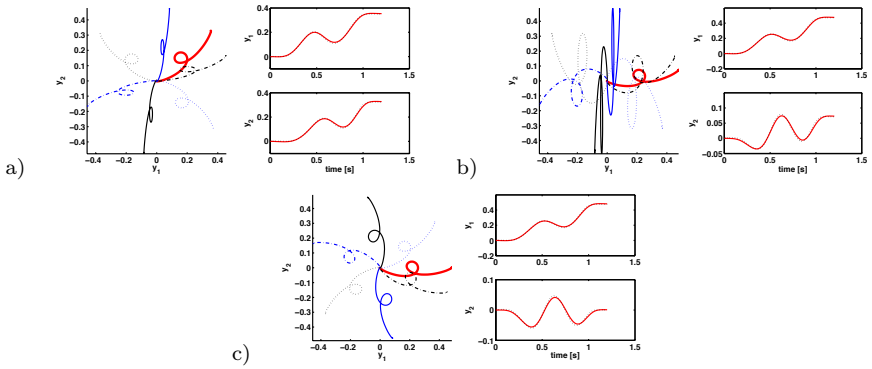


Fig. 2.5: Generalization of a 2 DOF discrete dynamical system under different choices of coordinate systems. The 2D movement is a point to point movement with a loop on the way to the goal. All movements start at the origin of the coordinate system and terminate at six different goal positions, distributed with 60 degree distance on a circle. The heavy (red) path in the first quadrant of the coordinate system was the originally learned movement. The generalization of this movement to 6 different targets is drawn with different line styles, to make it easier to see the paths of these movements. The two plots on the right of each subfigure show the y_1 and y_2 trajectories of each original movement. a) the original movement is in a benign part of the Cartesian coordinate system, b) again a Cartesian coordinate system, but the y_2 coordinate of the original movement has the start and end point of the movement within a small distance, c) choosing a coordinate system that has as the first coordinate the line between start and end point, and the second coordinate is perpendicular.

where y_a is the output feedback (e.g. exact robot's joint angles perceived by proprioception). The first equation is just a low pass filter of the tracking error $e = y_a - y$. This error is used as an additive coupling term in the transformation system, which hinders the state y to evolve too far away from y_a. Equation 2.17 affects the time constant τ of all differential equations of the dynamical system, i.e., both the canonical and the transformation system. This modification of the time constant slows down the temporal evolution of the dynamics in case of a significant tracking error.

Figure 2.6 illustrates the behavior due to these coupling terms (facing temporal perturbation for $t \in [0.35s, 0.9s]$) in comparison to the unperturbed (dashed line) time evolution of the dynamics. The top left plot of Figure 2.6 also shows with the dash-dot line the position y_a. During the holding time period, the entire dynamics comes almost to a stop, and resumes after the release of the mass roughly with the same behavior as where the system had left off.

2.4.11 Stability

Stability of the proposed dynamical systems equations can be examined on the basis that equation 2.1 is (by design) a simple 2nd order time-invariant linear system driven by a forcing term. The time constants of equation 2.1 are assumed to be chosen such that without the forcing term, the system is critically damped. Re-arranging equation 2.1 to combine the goal g and the forcing term f in one expression results in

$$\begin{aligned} \tau \dot{z} &= \alpha_z \beta_z \left(\left(g + \frac{f}{\alpha_z \beta_z} \right) - y \right) - \alpha_z z = \alpha_z \beta_z (u - y) - \alpha_z z \qquad (2.18) \\ \tau \dot{y} &= z \end{aligned}$$

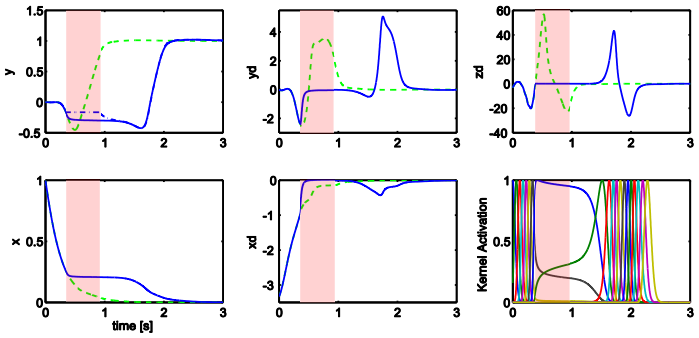


Fig. 2.6: Subjecting the discrete dynamical system from Figure 2.1 to "holding" (temporal) perturbation. At time $t = 0.35s$, the actual movement system is blocked from its time evolution, i.e., its velocity and acceleration are zero and its position (dash-dot line in the top-left figure) remains constant until $t = 0.9s$ (see shaded area). Due to the coupling terms, the time evolution of the dynamical system decays to zero, and resumes after the actual system is released. The unperturbed time evolution of the dynamics is shown in a dashed line, for comparison. Essentially, the perturbation simply delays the time evolution of the dynamical system.

where u is a time-variant input to the linear spring-damper system. Equation 2.18 acts like a low pass filter on u. For such linear systems, with appropriate α_z and β_z, e.g., from critical damping as employed in our work, it is easy to proof bounded-input-bounded-output (BIBO) stability, as the magnitude of the forcing function f is bounded by virtue that all terms of the function, i.e., basis functions, weights, and other multipliers are bounded by design. Thus, both the discrete and rhythmic system are BIBO stable. For the discrete system, given that f decays to zero, u converges to the steady state g after a transient time, such that the system will asymptotically converge to g. After the transient time, the system will exponentially converge to g as only the linear spring-damper dynamics remain relevant. Thus, ensuring that our dynamical systems remain stable is a rather simple exercise of basic stability theory. There is also another path to prove stability based on contraction theory; see [1].

2.5 Non-Functional Analysis

2.5.1 Representation and Interface

The input and output data of this system can be in global or local coordinate systems. The choice of the coordinate system will affect the outcome of different modulations (goal, amplitude, etc). The input of this model can have any dimensions and it is not dependent to any metric units. The input can be a trajectory of position, velocity or acceleration. The output of the system is the first derivatives of the state variables.

Trajectories are modeled as linear combination of phase-driven basis functions. The number of the basis functions are important if a precise output is needed. More basis functions is needed to model rapidly changing trajectories, and less ones to have smoother outputs (when having lots of noise). The number of the basis functions are fixed after the training phase is started (this is the case with standard LWR and can be replaced with locally weighted learning).

Finally, the approach is suitable to be boxed (with inputs and outputs). This means that there are well-defined, separate, and accessible variables as the inputs and

outputs of the system. Even modulation commands (e.g. τ or g) can be given to the system as inputs.

2.5.2 Timing

The phase variable of the canonical system works as the internal clock for coordination. This internal clock controls the coordination of subsystems. As another timing issue, as the outputs of the system are state derivatives, an integration timestep is used. Regarding this, as the computations are light, small Δt values can be considered to fulfill real-time constraints.

2.5.3 Robustness and Reliability

The introduced system can give anytime guarantee, i.e. it is able to respond to queries at any time during execution. As the load of the computations are low, satisfying real-time constraints is not a problem and the system is reliable in this regard. The model has some tunable parameters, and changing them will not affect the geometry of the attractor landscape (τ, g and r). So the model has invariance properties that preserves homomorphism.

2.5.4 Dependencies

The working of the system is dependent to the initialization of τ, g and r. The system is not dependent to any exogenous inputs. It can also work in an open-loop manner. However, is the open-loop case is of interest, the performance is dependent to a good controller (position or velocity or acceleration). Proprioception can be added to the system as a sensory feedback. This feedback is added to the model with a coupling term.

2.5.5 Runtime

This system can have different stages: offline learning, online learning, evaluation, recognition (with the help of an external tool). Offline learning, or each step of the online learning is one-shot and there is not any computation loops. The evaluation is also simple and it is just a time integration on the outputs of canonical and transformation systems. The runtime of the recognition part (that is just an extra feature) is totally dependent to the external tool used for classification.

2.5.6 Usefulness for Recognition

Due to the temporal and spatial invariance of this system's representation, an interesting aspect of this dynamical systems approach arises as trajectories that are topologically similar are fit by similar parameters w_i. This property opens the possibility of using our representation for movement recognition. It should be noted that such recognition is about spatiotemporal patterns, not just spatial patterns. Moreover, it is important to state that the recognition is done by an external tool, and the system only provides the data for recognition.

The ability to be useful for recognition is an advantage for an adaptive module. If new modules are learned in a hierarchical and stepwise manner, it would be useful if a tool for measuring the similarity of the adaptive modules and the underlying movements exist.

2.6 Summary

In this chapter, a computational model for Dynamical Movement Primitives (DMPs) is presented. The proposed system is based on a transformation system, i.e. a stable well-defined second-order spring-damper system, excited by a nonlinear forcing term. The forcing term is defined by Locally Weighted Regression (LWR) that can model any arbitrary input. The transformation system is driven by a canonical system that represents phase. Canonical system can be a simple point attractor system for discrete trajectories, and a simple phase oscillator for periodic ones. The proposed system is able to learn desired discrete or periodic trajectories in both offline and online manners.

Chapter 3

Stable Estimator of Dynamical Systems

S.Mohammad Khansari-Zadeh, Seungsu Kim, Aude Billard
EPFL, LASA

3.1 Short Introduction

Stable Estimator of Dynamical Systems (SEDS) provides a generic model to estimate any arbitrary multi-dimensional autonomous (i.e. time-invariant) nonlinear Dynamical System from a set of demonstrations of a task shown to a robot by a user [5–7]. SEDS optimizes the model of demonstrated motions based on different objective functions (e.g. likelihood, mean square error, etc.) under the constraint of the model's global asymptotic stability. Hence the obtained model is able to follow closely the demonstrations while ultimately reaching in and stopping at the target (see Figure 3.1).

The main advantage of learning DSs with SEDS is that it enables a robot to adapt *instantly* its trajectory in the faces of perturbations[1]. A controller driven by a DS is robust to perturbations because it embeds all possible solutions to reach a target into one single function (see Figure 3.1). Such a function represents a global map which specifies on-the-fly the correct direction for reaching the target, considering the current position of the robot and the target.

3.2 Model Description

3.2.1 SEDS Model Structure

We formulate the encoding of point-to-point motion as control law driven by an autonomous dynamical system: Consider a state variable $\xi \in \mathbb{R}^d$ that can be used to

[1] We consider two types of perturbations: 1) *spatial perturbations* which result from a sudden displacement in space of either the robot's arm or of the target, and 2) *temporal perturbations* which result from delays in the execution of the task. Note that we distinguish between spatial and temporal perturbations as these result in different distortion of the estimated dynamics and hence require different means to tackle these. Typically, spatial perturbation would result from an imprecise localization of the target or from interacting with a dynamic environment where either the target or the robot's may be moved by external perturbation; temporal perturbation arise typically when the robot may get stuck momentarily due to internal friction.

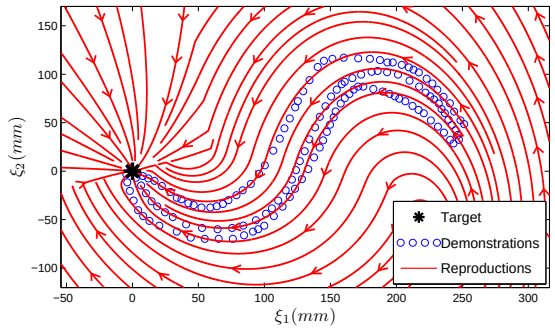


Fig. 3.1: Given a set of demonstrations, SEDS builds an estimate of the underlying dynamics of the motion such that it is globally asymptotically stable at the target while following accurately a specific motion with a particular dynamics.

unambiguously define a discrete motion of a robotic system (e.g. ξ could be a robot's joint angles, the position of an arm's end-effector in Cartesian space, etc). Let the set of N given demonstrations $\{\xi^{t,n}, \dot{\xi}^{t,n}\}_{t=0,n=1}^{T^n,N}$ be instances of a global motion model governed by a first order autonomous Ordinary Differential Equation (ODE):

$$\dot{\xi} = f(\xi) + \epsilon \tag{3.1}$$

where $f : \mathbb{R}^d \rightarrow \mathbb{R}^d$ is a nonlinear continuous and continuously differentiable function with a single equilibrium point $\dot{\xi}^* = f(\xi^*) = 0$, $\boldsymbol{\theta}$ is the set of parameters of f, and ϵ represents a zero mean additive Gaussian noise. The noise term ϵ encapsulates both inaccuracies in sensor measurements and errors resulting from imperfect demonstrations. The function $\hat{f}(\xi)$ can be described by a set of parameters $\boldsymbol{\theta}$, in which the optimal values of $\boldsymbol{\theta}$ can be obtained based on the set of demonstrations using different statistical approaches[2]. We will further denote the obtained noise-free estimate of f from the statistical modeling with $\hat{f}$ throughout the chapter. Our noise-free estimate will thus be:

$$\dot{\xi} = \hat{f}(\xi) \tag{3.2}$$

SEDS uses a probabilistic framework and models $\hat{f}$ via a finite mixture of Gaussian functions. Mixture modeling is a popular approach for density approximation [8], and it allows a user to define an appropriate model through a tradeoff between model complexity and variations of the available training data. Mixture modeling is a method, that builds a coarse representation of the data density through a fixed number (usually lower than 10) of mixture components.

Estimating f via a finite mixture of Gaussian functions, the unknown parameters of $\hat{f}$ become the prior π^k, the mean μ^k and the covariance matrices Σ^k of the $k = 1..K$ Gaussian functions (i.e. $\theta^k = \{\pi^k, \mu^k, \Sigma^k\}$ and $\boldsymbol{\theta} = \{\theta^1..\theta^K\}$). The mean and the covariance matrix of a Gaussian k are defined by:

$$\mu^k = \begin{pmatrix} \mu_\xi^k \\ \mu_{\dot{\xi}}^k \end{pmatrix} \quad \& \quad \Sigma^k = \begin{pmatrix} \Sigma_\xi^k & \Sigma_{\xi\dot{\xi}}^k \\ \Sigma_{\dot{\xi}\xi}^k & \Sigma_{\dot{\xi}}^k \end{pmatrix} \tag{3.3}$$

Given a set of N demonstrations $\{\xi^{t,n}, \dot{\xi}^{t,n}\}_{t=0,n=1}^{T^n,N}$, each recorded point in the trajectories $[\xi^{t,n}, \dot{\xi}^{t,n}]$ is associated with a probability density function $\mathcal{P}(\xi^{t,n}, \dot{\xi}^{t,n})$:

[2] Assuming a zero mean distribution for the noise makes it possible to estimate the noise free model through regression.

$$\mathcal{P}(\xi^{t,n}, \dot{\xi}^{t,n}; \boldsymbol{\theta}) = \sum_{k=1}^{K} \mathcal{P}(k)\mathcal{P}(\xi^{t,n}, \dot{\xi}^{t,n}|k) \quad \begin{cases} \forall n \in 1..N \\ t \in 1..T^n \end{cases} \tag{3.4}$$

where $\mathcal{P}(k) = \pi^k$ is the prior and $\mathcal{P}(\xi^{t,n}, \dot{\xi}^{t,n}|k)$ is the conditional probability density function given by:

$$\mathcal{P}(\xi^{t,n}, \dot{\xi}^{t,n}|k) = \mathcal{N}(\xi^{t,n}, \dot{\xi}^{t,n}; \mu^k, \Sigma^k) = \frac{1}{\sqrt{(2\pi)^{2d}|\Sigma^k_\xi|}} e^{-\frac{1}{2}([\xi^{t,n}, \dot{\xi}^{t,n}]-\mu^k)^T(\Sigma^k)^{-1}([\xi^{t,n}, \dot{\xi}^{t,n}]-\mu^k)} \tag{3.5}$$

Taking the posterior mean estimate of $\mathcal{P}(\dot{\xi}|\xi)$ yields (as described in [9]):

$$\dot{\xi} = \sum_{k=1}^{K} \frac{\mathcal{P}(k)\mathcal{P}(\xi|k)}{\sum_{i=1}^{K} \mathcal{P}(i)\mathcal{P}(\xi|i)} (\mu^k_{\dot{\xi}} + \Sigma^k_{\dot{\xi}\xi}(\Sigma^k_\xi)^{-1}(\xi - \mu^k_\xi)) \tag{3.6}$$

The notation of Eq. 3.6 can be simplified through a change of variable. Let us define:

$$\begin{cases} A^k = \Sigma^k_{\dot{\xi}\xi}(\Sigma^k_\xi)^{-1} \\ b^k = \mu^k_{\dot{\xi}} - A^k\mu^k_\xi \\ h^k(\xi) = \frac{\mathcal{P}(k)\mathcal{P}(\xi|k)}{\sum_{i=1}^{K} \mathcal{P}(i)\mathcal{P}(\xi|i)} \end{cases} \tag{3.7}$$

Substituting Eq. 3.7 into Eq. 3.6 yields:

$$\dot{\xi} = \hat{f}(\xi) = \sum_{k=1}^{K} h^k(\xi)(A^k\xi + b^k) \tag{3.8}$$

First observe that $\hat{f}$ is now expressed as a non-linear sum of linear dynamical systems. Figure 3.2 illustrates the parameters of Eq. 3.7 and their effects on Eq. 3.8 for a 1-D model constructed with 3 Gaussians. Here, each linear dynamics $A^k\xi + b^k$ corresponds to a line that passes through the centers μ^k with slope A^k. The nonlinear weighting terms $h^k(\xi)$ in Eq. 3.8, where $0 < h^k(\xi) \leq 1$, give a measure of the relative influence of each Gaussian locally. Observe that due to the nonlinear weighting terms $h^k(\xi)$, the resulting function $\hat{f}(\xi)$ is nonlinear and is flexible enough to model a wide variety of motions. If one estimates this mixture using classical methods such as Expectation Maximization (EM) [10], one cannot guarantee that the system will be asymptotically stable. The resulting nonlinear model $\hat{f}(\xi)$ usually contains several spurious attractors or limit cycles even for a simple 2-D model.

Figure 3.3 illustrates an example of unstable estimation of a two dimensional nonlinear DS using three different regression techniques: Locally Weighted Projection Regression (LWPR) [11], Gaussian Process Regression (GPR) [12], and Gaussian Mixture Regression (GMR) [13]. Because all of the aforementioned methods do not optimize under the constraint of making the system stable at the attractor[3], they are not guaranteed to result in a stable estimate of the motion. In practice, they fail to ensure global stability of $\hat{f}$, and thus may converge to a spurious attractor or completely miss the target (diverging/unstable behavior). First graph shows the obtained results from LWPR. All trajectories inside the dashed black boundary converge to a spurious attractor. Outside this boundary, the velocity is always zero, hence there exists regions of spurious attractors where motions stop once they cross the boundary. While for GPR trajectories converge to the target in a narrow area close to demonstrations, they are mainly attracted to regions of spurious attractors in the most parts of the operational space. The third graph represents the stability analysis of the dynamics

[3]GMR and GPR optimize the likelihood that the complete model represents well the data. LWPR minimizes the mean-square error between the estimate and the data.

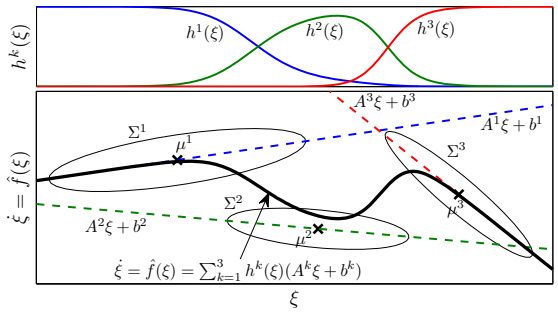


Fig. 3.2: Illustration of parameters defined in Eq. 3.7 and their effects on $\hat{f}(\xi)$ for a 1-D model constructed with 3 Gaussians. Please refer to the text for further information.

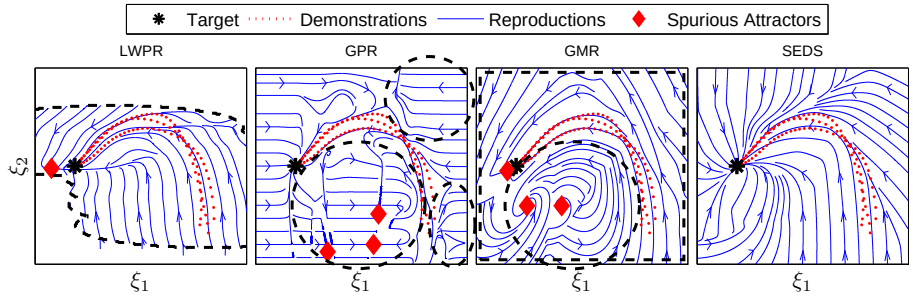


Fig. 3.3: Two dimensional dynamics estimated using (from left to right) LWPR, GPR, GMR and SEDS, respectively. Training was done using three demonstrations (red dots). Regions of the state space from which the system converges to spurious attractors are encircled by bold dashed lines.

learned with GMR. Here in the narrow regions around demonstrations, the trajectories converges to a spurious attractor just next to the target. In other parts of the space, they either converge to other spurious attractors far from the target or completely diverge from the target. In all these examples, regions of attractions are usually very close to demonstrations and thus should be carefully avoided. Most crucial is the fact that there is no theoretical solution to determine beforehand whether a trajectory will lead to a spurious attractor, to infinity, or to the desired attractor. Finding these regions of attraction is a non-trivial task and it becomes computationally costly in higher dimensions.

Next we determine sufficient conditions on the learning parameters $\boldsymbol{\theta}$ to ensure asymptotic stability of $\hat{f}(\xi)$.

3.2.2 SEDS Learning Algorithm

Section 3.2.1 provided us with a statistical formulation to model an arbitrary nonlinear function $\hat{f}(\xi)$. It remains now to determine a procedure for computing the unknown parameters of Eq. 3.8, i.e. $\boldsymbol{\theta} = \{\pi^1..\pi^K; \mu^1..\mu^K; \Sigma^1..\Sigma^K\}$ such that the resulting model is globally asymptotically stable. In this section we propose a learning algorithm, called *Stable Estimator of Dynamical Systems (SEDS)*, that computes the optimal values of $\boldsymbol{\theta}$ by solving an optimization problem under the constraint of ensuring the model's global asymptotic stability. We consider two different candidates for the optimization

objective function: 1) log-likelihood, and 2) Mean Square Error (MSE).

SEDS-Likelihood: *using log-likelihood as a means to construct a model.*

$$\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = -\sum_{n=1}^{N}\sum_{t=0}^{T^n} \log \mathcal{P}(\xi^{t,n}, \dot{\xi}^{t,n}|\boldsymbol{\theta}) \tag{3.9}$$

subject to

$$\begin{cases} \textbf{\textit{(a)}}\ b^k = -A^k \xi^T \\ \textbf{\textit{(b)}}\ \frac{1}{2}(A^k + (A^k)^T) < 0 \\ \textbf{\textit{(c)}}\ \Sigma^k > 0 \qquad\qquad \forall k \in 1..K \\ \textbf{\textit{(d)}}\ 0 < \pi^k \leq 1 \\ \textbf{\textit{(e)}}\ \sum_{k=1}^{K} \pi^k = 1 \end{cases} \tag{3.10}$$

where $\mathcal{P}(\xi^{t,n}, \dot{\xi}^{t,n}|\boldsymbol{\theta})$ is given by Eq. 3.4. The first two constraints in Eq. 3.10 are stability conditions taken from [5]. The last three constraints are imposed by the nature of the Gaussian Mixture Model to ensure that Σ^k are positive definite matrices, priors π^k are positive scalars smaller or equal than one, and sum of all priors is equal to one (because the probability value of Eq. 3.4 should not exceed 1).

SEDS-MSE: *using Mean Square Error as a means to quantify the accuracy of estimations based on demonstrations.*

$$\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \frac{1}{N}\sum_{n=1}^{N}\sum_{t=0}^{T^n} (\hat{\dot{\xi}}^n(t) - \dot{\xi}^{t,n})^2 \tag{3.11}$$

subject to the same constrains as given by Eq. 3.10.

In Eq. 3.11, $\hat{\dot{\xi}}^n(t) = \hat{f}(\hat{\xi}^n(t)$ are computed directly from Eq. 3.8.

Both SEDS-Likelihood and SEDS-MSE correspond to a Non-linear Programming (NLP) problem [14] that can be solved using different optimization techniques such as Newton and quasi-Newton algorithms [14], Dynamic Programming [15], etc. In the current work we use quasi-Newton method to solve the optimization problem. Quasi-Newton methods differ from classical Newton methods in that they compute an estimate of the Hessian function $H(\xi)$, and thus do not require a user to provide it explicitly. The estimate of the Hessian function progressively approaches to its real value as optimization proceeds. Among quasi-Newton methods, Broyden-Fletcher-Goldfard-Shanno (BFGS) is one of the most popular approaches for which there is a substantial evidence that it is the best general purpose quasi-Newton method currently known [14]. In the experiment presented in Section VI, we used the BFGS method using a line search method [16] to adaptively change the magnitude of each step to obtain an acceptable decrease in the objective function:

$$\boldsymbol{\theta}^{i+1} = \boldsymbol{\theta}^i + \alpha(\nabla\boldsymbol{\theta})^i \tag{3.12}$$

where α is the line-search's parameter, $(\nabla\boldsymbol{\theta})^i$ corresponds to the appropriate descent direction that minimizes the objective function under the given constrains, and i is the iteration step.

Note that a feasible solution to this NLP problem always exists. Starting from an initial value, the solver tries to optimize the value of $\boldsymbol{\theta}$ such that the cost function J is minimized. However since the proposed NLP problem is non-convex, one cannot ensure to find the globally optimal solution. Solvers are usually very sensitive to initialization of the parameters and will often converge to some local minimum of the objective function. We use the Bayesian Information Criterion (BIC) to choose the

optimal set K of Gaussians. BIC determines a tradeoff between optimizing the model's likelihood and the number of parameters needed to encode the data:

$$BIC = J(\boldsymbol{\theta}) + \frac{n_p}{2}\log(\sum_{n=1}^{N} T^n) \tag{3.13}$$

where $J(\boldsymbol{\theta})$ is the log-likelihood of the model computed using Eq. 3.9, and n_p is the total number of free parameters. The SEDS-Likelihood approach requires the estimation of $K*(2d^2+3d+1)$ parameters (the priors π^k, mean μ^k and covariance Σ^k are of size 1, $2d$ and $d(2d+1)$ respectively). However, the number of parameters can be reduced since the constraints given by Eq. 3.10-(a) provide an explicit formulation to compute $\mu^k_{\dot{\xi}}$ from other parameters (i.e. μ^k_{ξ}, Σ^k_{ξ}, and $\Sigma^k_{\dot{\xi}\xi}$). Thus the total number of parameters to construct a GMM with K Gaussians is $K(1+2d(d+1))$. As for SEDS-MSE, the number of parameters is even more reduced since when constructing $\hat{f}$, the term $\Sigma^k_{\dot{\xi}}$ is not used and thus can be omitted during the optimization. Taking this into account, the total number of learning parameters for the SEDS-MSE reduces to $K(1+\frac{3}{2}d(d+1))$. For both approaches, learning grows linearly with the number of Gaussians and quadratically with the dimension. In comparison, the number of parameters in the proposed method is fewer than GMM and LWPR[4]. The retrieval time of the proposed method is low and in the same order of GMR and LWPR.

3.3 Simulation

Figure 3.4 demonstrates the performance of SEDS in learning a library of human handwriting motions. These motions were recorded from a Tablet-PC, and each motion was demonstrated 3 to 5 times. Here, the models were learned using likelihood as the objective function for the optimization. As can be seen, the resulting SEDS models capture well the non-linearity of these complex motions.

3.4 Functional Analysis

3.4.1 Dynamics and Nonlinearity

The proposed method is able to model any arbitrary multi-dimensional non-linear motion. The method considers correlation between all dimensions, and is able to generalize across several demonstrations. SEDS guarantees the asymptotic stability of the learned model.

3.4.2 Attractor

SEDS converges to a point attractor. This point attractor could be set to be the final target for the system or it could be considered as a via point. In other words, SEDS could always guarantee convergence to any arbitrary desired target with or without passing through some given via points.

3.4.3 Coupling

SEDS considers a coupled system in modeling of a demonstrated task. The strength and coupling relations across dimensions are automatically learned during learning procedure.

[4]The number of learning parameter in GMR and LWPR is $K(1+3d+2d^2)$ and $\frac{7}{2}K(d+d^2)$ respectively.

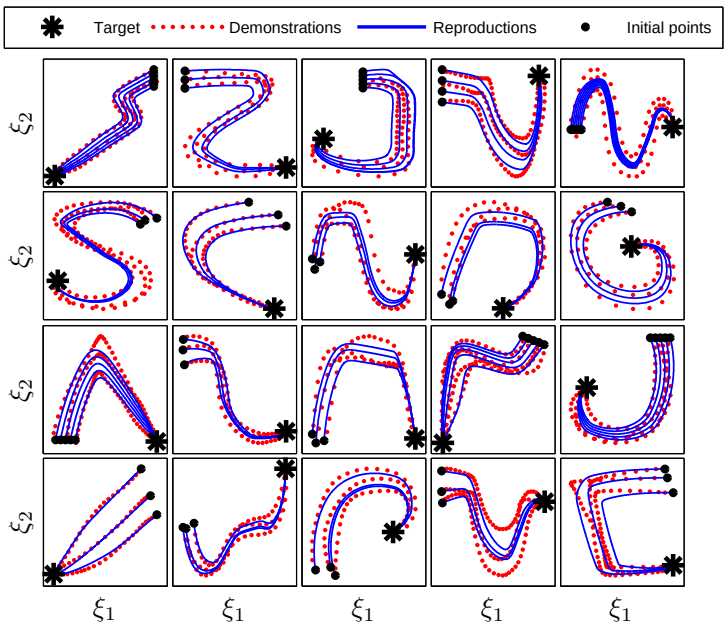


Fig. 3.4: Performance illustration of SEDS in learning 20 different human handwriting motions using likelihood as the objective function.

3.4.4 Learning

Type The type of learning is supervised by requiring a user to show to a robot a few demonstrations of the desired task.

Algorithm Stable Estimator of Dynamical System (SEDS). For more detailed information please refer to 3.2.2.

Mode Learning is offline.

3.4.5 Training Data

The input training data to the model is a few demonstrations of the task (it is defined with the variable ξ in Section 3.2.1). ξ is a set of variable that can be used to unambiguously define a discrete motion of a robotic system (e.g. ξ could be a robot's joint angles, the position of an arm's end-effector in Cartesian space, etc).

3.4.6 Generalization

SEDS is able to generalize a task in three different aspects

1. Generalization to doing a task from different initial positions (see Figure 3.1)
2. Generalization to the change in the target's position (see Figure 3.5). We would like to highlight this point that in an SEDS model, the generalization *is not done* by simply multiplying the original model with a scaling factor. Our model is able to generalize based on the dynamic of motion that is learned based on a couple of demonstrations. Regarding Figure 3.5, one can observe that the generalization to different position of the targets is done by changing only a small part of the motion (in contrast to scaling where the whole motion is affected).

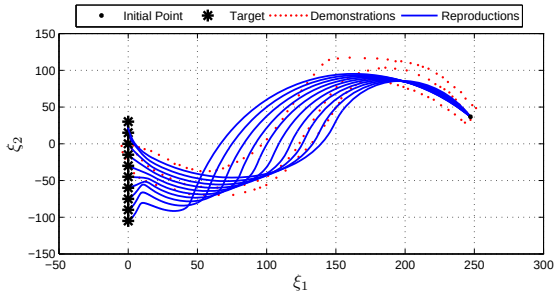


Fig. 3.5: Generalization to different positions of the target.

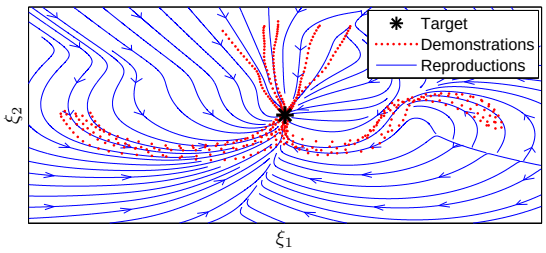


Fig. 3.6: Embedding different ways of performing a task in one single model. The robot follows an arc, a sine, or a straight line starting from different points in the workspace.

3. Generalization to different ways of doing a task. To elaborate more, assume that it is desired to do a same task in different manners starting from different areas in space, mainly to consider task constraints, to avoid robot's joint limits, etc. Figure 3.6 shows an example of such a task where a robot required to approach to the target following an arc, a sine, or a straight line path starting from the left, right, or top-side of the task space, respectively. While reproductions locally follow the desired motion around each set of demonstrations, they smoothly switch from one motion to another in areas between demonstrations. The proposed method offers a simple but reliable procedure to teach a robot different ways of performing a task.

3.4.7 Modulation

SEDS is able to smoothly modulate working parameters without affecting the geometry of the basin of attraction. The modulation can simply done by multiplying Eq. 3.8 by a desired factor.

$$\dot{\xi} = \lambda \hat{f}(\xi) \tag{3.14}$$

where λ is the modulation factor.

3.4.8 Sensory Feedback Integration

One of the main advantages of our proposed model is its simple integration to a real closed loop, hence re-correcting on-the-fly the next command based on the current state of the robot perceived through sensors. Figure 3.7 shows a schematic of the control flow for an arbitrary system using a SEDS model. The whole system's architecture

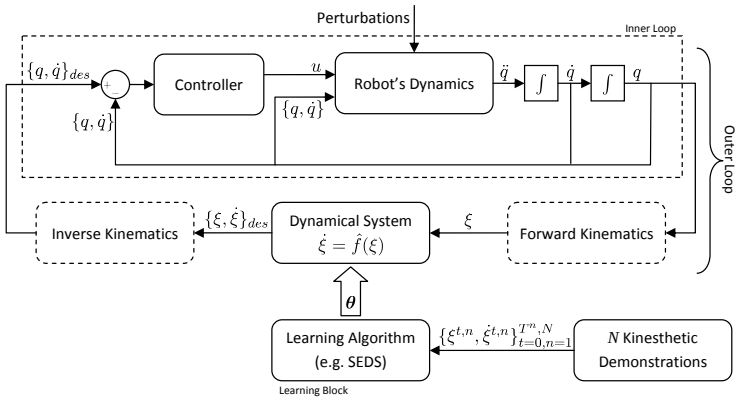


Fig. 3.7: A typical system's architecture illustrating the control flow in a robotic system as considered for SEDS. The system is composed of two loops: the inner loop representing the robot's dynamics and a low level controller, and an outer loop defining the desired motion at each time step. The learning block is used to infer the parameters of motion $\boldsymbol{\theta}$ from demonstrations.

can be decomposed into two loops. The inner loop consists of a controller generating the required commands to follow the desired motion and a system block to model the dynamics of the robot. Here q, $\dot{q}$, and $\ddot{q}$ are the robot's joint angle and its first and second time derivatives. Motor commands are denoted by u. The outer loop specifies the next desired position and velocity of the motion w.r.t. the current status of the robot. An inverse kinematics block may also be considered in the outer loop to transfer the desired trajectory from cartesian space to joint space (this block is not necessary if the motion is already specified in joint space). Having controlled by a SEDS model, the outer loop controller can handle the inner loop controller's inaccuracy by treating these as perturbations, comparing the expected versus the actual state of the system.

3.4.9 State Variables

An arbitrary motion is modeled with the d-dimensional state variable $\xi \in \mathbb{R}^d$ and its first time derivative $\dot{\xi}$.

3.4.10 Robustness and Adaptation to Perturbations

A SEDS model is inherently designed to be robust to any spatial or temporal perturbations (see Section 3.2). In SEDS, without loss of generality we assume the origin of the reference coordinate systems at the target. Any spatial perturbation can be perceived as a change in the state variable ξ w.r.t. this frame of reference. Since an SEDS model is globally asymptotically stable, the convergence to the target is thus always guaranteed. The proposed model is inherently time-independent, hence it is robust to any temporal perturbations. Besides the inherent robustness to perturbations, the proposed model has the capability of adapting a new motion *on-the-fly* in the face of perturbation without any need to replanning, re-indexing, or re-scaling.

3.4.11 Stability

A SEDS model is globally asymptotically stable. The stability conditions of the system is forced as constraints in the learning algorithm (see Eq. 3.10). For the proof of stability please refer to [5].

3.5 Non-Functional Analysis

3.5.1 Representation and Interface

The proposed model can be used in any coordinate systems and dimensions, and does not depend on the metric units. The motion is modeled as a non-linear time-independent function based on a Gaussian Mixture Model (GMM). The number of Gaussian functions in GMM can be defined either manually or based on Bayesian Information Criterion (BIC) [17] (see Eq. 3.13).

3.5.2 Timing

An integration time-step needed to be defined by the user, which will be used to integrate the first derivative of the state variable. The model is very fast and can handle the realtime constraints.

The accurate motion timing is highly important if a robot has to synchronize with external moving objects. As discussed in Section 6.4.7, our non-linear dynamical system can be extended to allow one to modulate the speed of the motion [18]. The modulation factor can be used to control the motion duration so as to speed up or slow down the robot's motion and hence adhere to precious temporal constraints, while still benefitting from all the robustness properties deriving from the time-independent encoding of the DS.

To allow for gradual and on the fly adaptation of the motion's duration so as to reach a position ξ^* in a given time T^*, we compute our multiplier at each time step as follows:

$$\xi^{t_{j+1}} = \xi^{t_j} + \lambda^{t_i} \sum_{l=1}^{L} \dot{\hat{\xi}}^{\{t_j + \frac{\Delta t}{L} l\}} \frac{\Delta t}{L} \tag{3.15}$$

$$\lambda^{t_{i+1}} = \lambda^{t_i} + k_p \left(\hat{T}^{t_i} - T^* \right) - k_d \left(\hat{T}^{t_i} - \hat{T}^{t_{i-1}} \right) \tag{3.16}$$

where t_i is a time at the i^{th} controlling step, $t_{i+1} = t_i + \Delta t$, $t_0 = 0$; λ^{t_i} is the velocity multiplier, $\lambda^{t_0} = 1$; k_p and k_d are user defined proportional and derivative gains that control for the reactivity of the system; $\hat{T}^{t_i}$ is the estimated overall motion's duration (starting from the beginning of the motion at time t_0) as calculated at time t_i; This duration $\hat{T}$ of the motion is estimated by integrating Eq. 3.14 until reaching the attractor [5].

3.5.3 Robustness and Reliability

The introduced system can give anytime guarantee, i.e. it is able to respond to queries at any time during execution. As the load of the computations are low, satisfying real-time constraints is of no problem and the system is reliable in this regard. As discussed in Section 3.4.10, the model is robust to change in the goal, modulation, etc. The optimization presented in Section 3.2.2, requires an initial guess for the model. The result of optimization is invariant w.r.t. small changes in the initial model's guess; however, large change may result to convergence into another optimal model. In any case the final model is globally asymptotically stable and robust to any perturbations.

[5] To reduce the negative effect of a big integration step, we integrate the dynamical law $\hat{f}$ L times, before sending an actual command to the robot

3.5.4 Dependencies

The working of the system is dependent to the number of Gaussian functions and the initial guess for the model. The system is not dependent to any exogenous inputs. It can also work in an open loop manner. However, if the open-loop case is of interest, the model's performance directly depends on the performance of the controller. Proprioception can be added to the system as a sensory feedback (See Figure 3.7)

3.5.5 Runtime

The system has three stages: offline learning, evaluation, and recognition. Learning is based on a non-linear optimization of a cost function, and thus works in a loop until convergence to the optimal point. Evaluation of any point can be done in one shot without any need to integrate. However, to generate a path from its initial point to the target one needs to integrate from Eq. 3.8, which can be done on-line. Since SEDS models a motion based on a statistical framework (i.e. GMM), recognition can also be using the learned statistical model (please see next the section for more details).

3.5.6 Usefulness for Recognition

SEDS learns a motion by fitting a GMM model onto demonstration datapoints. This GMM model can also be used as a statistical tool for movement recognition. For a given movement, one can simply compute the likelihood of that movement using an existing library of motion primitives. The movement recognition can be used for both spatial and spatial-temporal patterns; however, for cases where movements are spatially separable (see Figure 3.6), SEDS by itself can embed these movements into one single model, hence facilitating the recognition phase.

3.6 Summary

In this chapter we presented a method called Stable Estimator of Dynamical Systems (SEDS) that allows for fast learning of robot motions from a small set of demonstrations. It considers a motion as time-independent nonlinear dynamical systems and formulates it as a mixture of Gaussian functions. The parameters (the priors, centers, and covariances) of the GMM are learned via an optimization under constraints on the global asymptotic stability of the model. The main features of a model learned by SEDS can be summarized as follows:

- It guarantees the global asymptotic stability of motions.
- It is inherently robust to external perturbations.
- It is able to on-the-fly generate a new motion in the face of perturbations without any need to re-planning, re-indexing, or re-scaling.
- It can consider via points along the path to the target (passing through via points is always guaranteed.)
- It can generalize a motion to areas not covered before.
- It can embed different ways of performing a task into one single model.
- It is can easily modulate a model without affecting the main properties of the model.
- It can manipulate a task in a specified time duration.
- It can be used for the recognition of the movement.

Chapter 4

Neural Dynamical Motion Primitives Generator

Tim Waegeman, Benjamin Schrauwen
UGent

4.1 Short Introduction

Neural Dynamical Motion Primitives Generator (NDMPG) is an adaptive module which uses the Reservoir Computing (RC) technique to train its internal recurrent neural network (RNN). This generator is able to learn different motion trajectories in a single dynamical system. A low pass filter is integrated to allow velocity modulation of the generated motion. Additionally, it is possible to avoid obstacles by using a method which uses a nonlinear coordinate transformation. Depending on the used conditioning, the resulting trajectories show similarities with the trajectory of an end effector in a force field which is subject to a repelling force from the origin of the obstacle.

4.2 Model Description

The basic idea behind the Neural Dynamical Motion Primitives Generator is to use a recurrent neural network to embed the desired attractor. The generated trajectories are modulated afterwards to allow different velocities of the generated motion.

A recurrent neural network (RNN) contains neurons which are interconnected with each other. The state of each neuron is represented by $x_i[k] \in \mathbf{x}[k]$ at time step k which is updated according to:

$$\mathbf{x}[k+1] = \tanh\left(\mathbf{W}_{\mathrm{r}}^{\mathrm{r}}\mathbf{x}[k] + \mathbf{W}_{\mathrm{o}}^{\mathrm{r}}\mathbf{y}[k]\right), \tag{4.1}$$

where the connections weights between the neurons are given by $\mathbf{W}_{\mathrm{r}}^{\mathrm{r}}$. The weights $\mathbf{W}_{\mathrm{i}}^{\mathrm{r}}$ are scaled by a parameter called 'spectral radius' (ρ) to insure[1] the stability of the reservoir. $\mathbf{W}_{\mathrm{o}}^{\mathrm{r}}$ contains the feedback connections from the output layer to the

[1] depends on the value of the spectral radius. The connection weights are divided by the maximal absolute eigen value and multiplied by the spectral radius. Consequently, if the spectral radius is chosen smaller than one, thus operating in the non-chaotic domain, stability can be insured. However, if it is chosen larger than one the reservoir operates in the chaotic domain.

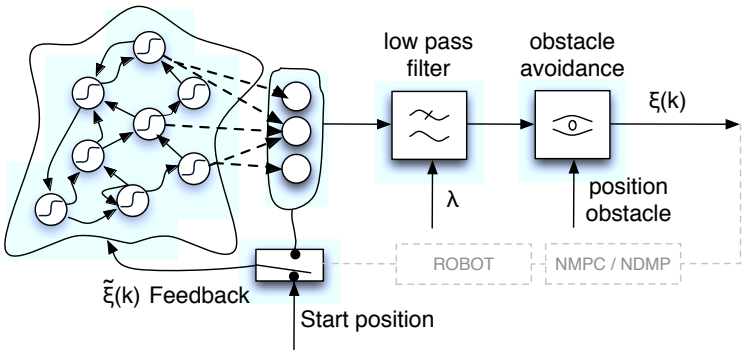


Fig. 4.1: Illustration of the proposed Neural Dynamical Motion Primitives Generator. The trained trajectories are embedded in the reservoir. The feedback $\tilde{\xi}(k)$ is initialized with the initial position (taking translation of the target position into account) after which the reservoir starts generating the desired trajectory. The target position is always $\mathbf{0}$. This means that the generated trajectory has to be translated when an other target position is required. The modulation of the velocity (parameter λ) is acquired by a low pass filter. Finally, the trajectory is adjusted when close to an obstacle ($\xi(k)$). When commanding a robot, the total controlling system including the robot and a controlling method like NDMP or NMPC, should be a part of the feedback

reservoir. As there is no input used, one can notice the absence of an input vector. As non-linearity a $tanh()$-function is used.

The output of the RNN is computed by the following equation:

$$\mathbf{y}[k+1] = \mathbf{W}_{\mathrm{r}}^{\mathrm{o}}\mathbf{x}[k+1], \tag{4.2}$$

where $\mathbf{W}_{\mathrm{r}}^{\mathrm{o}}$ represent the connections between the RNN and the output layer.

We will use the Reservoir Computing (RC) approach [19–21] where only the output weights $\mathbf{W}_{\mathrm{r}}^{\mathrm{o}}$ are trained, the other weights are randomly chosen. Under this approach, the used RNN is called 'the reservoir'.

The training of the output weights can be computed in one shot, according to the echo state approach [19]:

$$\mathbf{W}_{\mathrm{r}}^{\mathrm{o}} = \mathbf{T}\mathbf{X}^{T}(\mathbf{X}\mathbf{X}^{T} + \gamma\mathbf{I})^{-1}, \tag{4.3}$$

where matrix $\mathbf{X}$ consists of all reservoir states $\mathbf{x}[k]$. Matrix $\mathbf{T}$ consists of all desired outputs (teacher forced outputs). γ is the regularisation parameter. To improve generalisation capabilities and to prevent overfitting of the data, ridge regression is used. The regularization parameter γ is optimized on a validation set. For testing and global evaluation of the system, the optimal regularization parameter is used.

In Table 4.1 we summarize the used training parameters.

As shown in Figure 4.1, the reservoir uses output feedback which is initialized after training with the initial position. However, depending on the desired behaviour it is also possible to feed $\xi(k)$ back to the reservoir (whether or not with robot feedback). The RNN will generate a trajectory from this initial position to $\mathbf{0}$. Nevertheless, reaching to a different target position is possible after a simple translation of the trajectory. Consequently, the initial position given to the RNN should be translated as well.

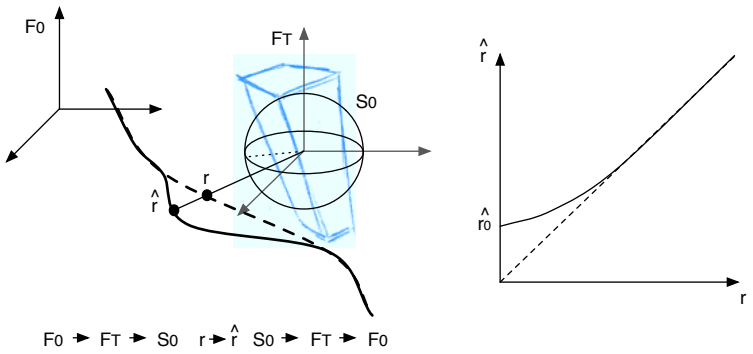


Fig. 4.2: Left illustration shows the transformation sequence to avoid an obstacle. F_0 denotes the original Cartesian frame of reference, F_T is the translated frame of reference and S_0 is a transformation of F_T to a spherical coordinate system. The bold curve on the left shows the adjusted trajectory while the dashed one shows the original trajectory. $\widehat{r}$ is the conditioned radius r with $\widehat{r}_0$ the minimum radius. Depending on this conditioning one can achieve similar repelling behaviour in a force field as setting a repelling force at the center of the obstacle. The right plot illustrates a possible conditioning on r.

parameter	description	value
N	reservoir size	$\in [100, 1000]$
ρ	spectral radius	0.9
ς	connection fraction RNN	100%
η	feedback scaling	1

Tab. 4.1: Training parameters for the NDMPG

To regulate the velocity a low pass filter is used:

$$\tilde{\mathbf{y}}[k+1] = (1-\lambda)\tilde{\mathbf{y}}[k] + \lambda \mathbf{y}[k], \tag{4.4}$$

where $\tilde{\mathbf{y}}[k]$ and $\mathbf{y}[k]$ describe the filter output and input, respectively. λ is the time constant which can be adjusted to modulate the velocity.

To incorporate the possibility to avoid obstacles during trajectory generation, each trajectory point in its original frame of reference F_0 is translated to the frame of reference F_T which is located at the position of the obstacle. Next, the current trajectory point is transformed from a Cartesian to a spherical coordinate system S_0. In this representation (r, θ, ψ) the radius r is conditioned to $\widehat{r}$ with a certain minimum distance $\widehat{r}_0$ as parameter. Depending on this conditioned radius, one can generate trajectories similar to a repelling point in a force field during force control. Afterwards, the total transformation is reversed. As shown in Figure 4.2 the resulting trajectory will avoid the predefined obstacle.

4.3 Simulation

The results of some small experiments are presented. Figure 4.3 shows the modulation effect of the used low pass filter with different time constants which demonstrates the

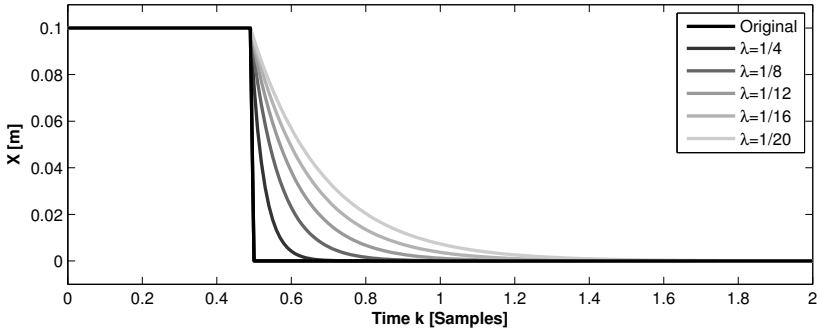


Fig. 4.3: Illustration of an experiment during which the velocity of the position information is modulated for different values of λ.

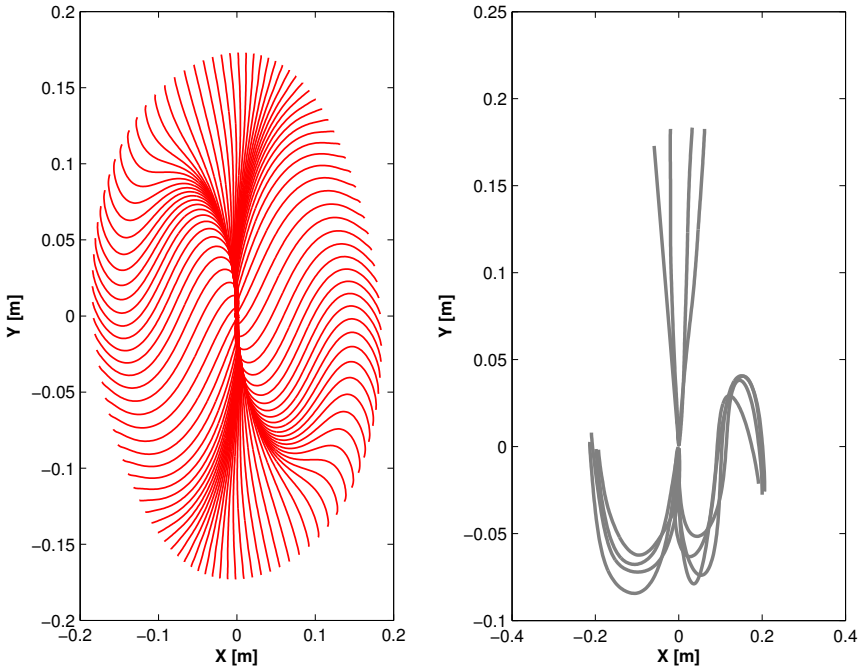


Fig. 4.4: The left plot shows the results of the generalisation capabilities of the NDMPG. In this experiment the reservoir was set to generate trajectories of different starting positions located on a circle. The plot on the right illustrates the examples which are used to train the reservoir. By comparing both plots one can notice the symmetry of the generated trajectories which were unseen during training. Additionally, we can conclude that an NDMPG is able to generate different trajectories depending on its starting position. When its starts at the right it can generalize an 'S'-motion and when it start at the top, it generalizes an 'I'-motion. However, the generalisation of the 'C'-motion in this case is not so good. This can be improved by increasing the amount of examples during training.

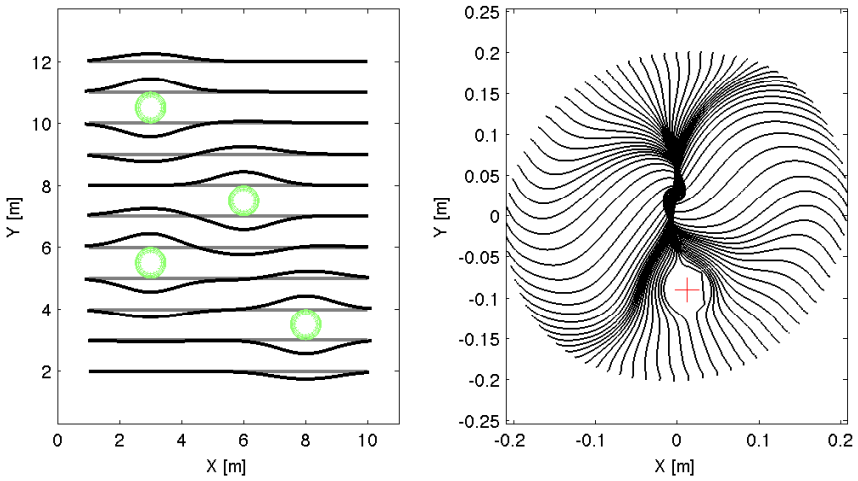


Fig. 4.5: Both plots illustrate the results of different experiment during which trajectories need to avoid obstacles.In the left plot, one can notice how example trajectories (black lines) avoid the introduced obstacles (circles). The trajectories without obstacles are illustrated as well (gray lines). The right plot illustrates obstacle avoidance of generalized trajectories. In this plot, the position of the obstacle is depicted by a cross.

spatiotemporal invariance. Figure 4.4(left) illustrates the generalisation capabilities of the proposed method. Figure 4.4(right) shows the used training examples. Figure 4.5 depicts the obstacle avoiding behaviour for different trajectories. A more detailed discussion about these results will be given later in this chapter.

4.4 Functional Analysis

4.4.1 Dynamics and Nonlinearity

An NDMPG integrates a reservoir which contains neurons featuring a $tanh()$ as nonlinearity. The system dynamics are the result of the interaction between those neurons and are therefore nonlinear as well. A $tanh()$ has a linear region around 0, so depending on the absolute feedback values, the dynamics can be more linear. If the spectral radius ρ is smaller than 1 the RNN operates in a stable regime. However, when $\rho > 1$, it is operating in a chaotic regime. Because of this, the nonlinear dynamics depend on the feedback and the chosen spectral radius.

4.4.2 Attractor

The system's attractor type depends on the training data given. In our experiments we used training trajectories which induce a point attractor into the reservoir with possible multiple basins of attraction (always converging to $\mathbf{0}$).

4.4.3 Coupling

The system generates end-effector (EE) trajectories. As mentioned before NDMPG's should be used in a hybrid system together with a controlling method like Neural

Motion Primitive Control (NMPC). The coupling between each Degree-of-Freedom (DoF) is managed by such method. In a controlling method like NMPC, the angles are generated by one dynamical system which means that the hybrid system is coupled.

4.4.4 Learning

Type The learning type of the system is supervised.

Algorithm The echo state approach of learning is used. The learning is done in one shot according to equation 4.3.

Mode The learning is achieved offline.

4.4.5 Training Data

The containing RNN is trained offline with example data. A limit set of example trajectories is sufficient. However, to achieve good generalisation capabilities, a large amount of training data is recommended.

4.4.6 Generalization

In Figure 4.4(left), the generalisation capabilities, in the sense of generating similar trajectories as seen during training, are shown. On the right of Figure 4.4 the used training examples are shown. If we compare the training data with the generated trajectories it is clear that the attractors within one RNN are shaped to reproduce all the training data as well as possible. As a result, symmetry in the dynamical system emerges. For example, reaching motions from the bottom to the center are similar to trained motions from the top to the center.

4.4.7 Modulation

The generated motion of an NDMPG can be modulated to allow different velocities of the motion. This is achieved by the introduced low pass filter of the system with time constant λ. In Figure 4.3 this behaviour is demonstrated for different λ's. Additionally, as this generator should be used in a hybrid system, the velocity can be modulated in the controlling system as well (like the leak rate in NMPC).

4.4.8 Sensory Feedback Integration

As mentioned before the proposed system uses feedback. When controlling a robot, the actual end-effector position should be fed back into the reservoir. This is illustrated in Figure 4.1.

4.4.9 State Variables

The amount of neurons is equal to the amount of state variables.

4.4.10 Robustness and Adaptation to Perturbations

Because of the designed system and its single point attractor, the reaching motion started from any position will eventually converge to the desired target position. As the system is designed to generate a reaching motion by initializing the feedback it is possible to perturb this feedback by any value as long as needed (e.g. keeping the feedback constant). After removing the perturbation the motion will converge back to its single point attractor. This behaviour is similar to the Dynamical Movement Primitives (DMP) described in an other chapter.

parameter	description	value
N	reservoir size	$\in [100, 1000]$
$\xi(k)$	desired position	X,Y and Z in (m)
$\tilde{\xi}(k)$	feedback	X,Y and Z in (m)
λ	time constant	$\in [10^{-6}, 1]$
$\widehat{r}$	minimum obstacle distance	in (m)

Tab. 4.2: Interface parameters

As shown in Figure 4.4(left), a perturbation which causes the end-effector to move to another position in the neighbourhood, will generate a similar converging motion.

4.4.11 Stability

Because of the used nonlinearity, the neuron states are bounded to $[-1, 1]$. The output is a linear mapping of these neuron states. This means that the system output is bounded as well. Although in most cases the desired attractor is reached, convergence can not be proven.

4.5 Non-Functional Analysis

4.5.1 Representation and Interface

The proposed generator has only one output and feedback. The desired end-effector position as output and the actual position as feedback. The end effector position needs to be scaled down to meters to insure no neuron state is saturated. Other parameters which are controllable are the time constant λ, the initial position (as feedback) and the position of an obstacle. Additionally, it is possible to define a certain minimum distance $\widehat{r}$ between the trajectory and the obstacle.

In Table 4.2 we summarize the interface parameters.

4.5.2 Timing

The proposed motion generator has an upper time limit. During operation only equations 4.1 and 5.2 need to be calculated. Both calculations are simple matrix multiplications and are limited in calculation time. Because offline training is used, the one shot calculation (equation 4.3) does not need to be recalculated during operation.

4.5.3 Robustness and Reliability

Depending on the training data and the number of different example motions the convergence to $\mathbf{0}$ is achieved. Because of its integrated low pass filter, the corresponding converging time can be modulated. Due to the limited calculations, a good Real-time constraint is possible.

4.5.4 Dependencies

The proposed motion generator needs the actual end-effector position as feedback. Both proprioception and exterioception can be used to determine this position. Furthermore, the system is implemented in MatLab.

4.5.5 Runtime

As indicated before the proposed adaptive module uses offline training where only equations 4.1 and 5.2 need to be computed during testing. These calculations do not request a lot of resources.

4.5.6 Usefulness for Recognition

As shown in Figure 4.4(left) the generalisation of the trained motions shows symmetry when using different initial positions. This identification can be used to identify similarities within different training motions as the dynamical system tries to capture them all in a single pool of neurons (reservoir). This identification is performed subjectively by evaluating the results and is not quantized by any parameter of the system.

4.6 Summary

In this chapter, we described an adaptive module called Neural Dynamical Motion Primitives Generator (NDMPG) which uses a recurrent neural network (RNN) to learn different trajectories. The initial position of the trajectory is set by initializing the feedback of the system. These generated trajectories can be modulated in time to regulate the velocity by using the integrated low pass filter. Additionally, we described a method to do obstacle avoidance while generating a trajectory. This shows similarities with the effect of putting a repelling force at the position of the object while performing force control of the end-effector. Finally, the results of some simple experiments are shown together with a detailed description of its characteristics.

Chapter 5

Neural Motion Primitive Control

Tim Waegeman, Benjamin Schrauwen
UGent

5.1 Short Introduction

Neural Motion Primitive Control (NMPC) is a closed loop control strategy which controls a dynamical system[1]. Hereby the Reservoir Computing technique is used to train a recurrent neural network (RNN). This RNN learns an internal model of the dynamical system. In addition, the proposed system is designed so that it can adjust its internal model during operation and adapt to unforeseen changes. Each degree-of-freedom (DoF) of a robot, is generated by the same RNN. The velocity and shape of the generated motion can be modulated by a set of parameters. To improve generalisation capabilities this system can use any trajectory generating solution (like NDMPG, Neural Dynamical Motion Primitives Generator) in a hybrid configuration.

5.2 Model Description

The concept behind this model is to construct an inverse model of, for example, a robot arm. Therefore, the robot will follow a desired trajectory and thus generating a reaching motion.

For modeling the dynamical system we use a recurrent neural network (RNN) with leaky integrator neurons. The state of each neuron is represented by $x_i[k] \in \mathbf{x}[k]$ at time step k. The update function of each state is given by:

$$\mathbf{x}[k+1] = (1-\lambda)\mathbf{x}[k] + \lambda \tanh\left(\mathbf{W}^{\mathrm{r}}_{\mathrm{r}}\mathbf{x}[k] + \mathbf{W}^{\mathrm{r}}_{\mathrm{i}}\mathbf{u}[k] + \mathbf{W}^{\mathrm{r}}_{\mathrm{o}}\mathbf{y}[k] + \mathbf{W}^{\mathrm{r}}_{\mathrm{b}}\right), \tag{5.1}$$

where the connections weights between the neurons are given by $\mathbf{W}^{\mathrm{r}}_{\mathrm{r}}$, $\mathbf{W}^{\mathrm{r}}_{\mathrm{i}}$ represents the connections from the input layer to the RNN. $\mathbf{W}^{\mathrm{r}}_{\mathrm{b}}$ represents the connections from the bias to the RNN. The weights $\mathbf{W}^{\mathrm{r}}_{\mathrm{i}}$ are scaled by a parameter called 'spectral

[1] NMPC is different from other chapters because it learns (unsupervised) an internal model of the robot arm. Additionally, by using the transients of the network and a large amount of training data, it is possible to regenerate (generalize) a trained trajectory.

radius' (ρ) to insure[2] the stability of the reservoir. $\mathbf{W}_{\text{o}}^{\text{r}}$ contains the connections from the output layer to the reservoir. The input vector is given by $\mathbf{u}[k]$. A $tanh()$-function is used as non-linearity. The parameter λ represents the leak rate which induces, when different from 1, a lowpass filter over the neuron states. As a result, the RNN has fading memory.

The output of the RNN is computed by the following equation:

$$\mathbf{y}[k+1] = \mathbf{W}_{\text{r}}^{\text{o}}\mathbf{x}[k+1], \tag{5.2}$$

with $\mathbf{W}_{\text{r}}^{\text{o}}$ the connections between the RNN and the output layer.

We will use the Reservoir Computing (RC) approach [19–21] where only the output weights $\mathbf{W}_{\text{r}}^{\text{o}}$ are trained, the other weights are randomly chosen. Under this approach, the used RNN is called 'the reservoir' and the total system (input, reservoir and output), the 'RC-network' (RCN).

The necessary output weights can be determined offline or online. The first procedure, needs recorded example input-output pairs of the desired behaviour over a long time. The later, learns while controlling, and can learn from examples, although optional. In this chapter we will use the term 'offline learning mode' when the learning algorithm is suspended after the training phase. The online learning mode, on the other hand, will never suspend the learning algorithm. Online and offline training is achieved by using recursive least squares (RLS), similar to the FORCE method [22]. After random initialisation of the output weights, these weights are updated (with $\Delta\mathbf{W}$) each time step. This update is according to an error $\mathbf{e}$ and an approximation of the inverse of the reservoir states correlation matrix ($\mathbf{P}$). The error $\mathbf{e}$ between the produced and the desired output at time step k is given by:

$$\mathbf{e} = \mathbf{W}(k-1)\mathbf{x}(k) - \mathbf{d}(k), \tag{5.3}$$

where $\mathbf{d}(k)$ is the desired output. The weight update is done according to the following equations:

$$\mathbf{P}(0) = \frac{\mathbf{I}}{\alpha}, \tag{5.4}$$

$$\mathbf{P}(k) = \mathbf{P}(k-1) - \frac{\mathbf{P}(k-1)\mathbf{x}(k)\mathbf{x}^T(k)\mathbf{P}(k-1)}{1+\mathbf{x}^T(k)\mathbf{P}(k-1)\mathbf{x}(k)}, \tag{5.5}$$

$$\Delta\mathbf{W}(k) = \mathbf{W}(k) - \mathbf{W}(k-1) = -\mathbf{e}\mathbf{P}(k)\mathbf{x}(k), \tag{5.6}$$

where α determines the learning rate of the algorithm.

The proposed model, illustrated in Figure 5.1 consists of 2 identical RC-networks. The first network, 'RCN1', gets the current end effector (EE) position $\xi(k)$ as input and a delayed version $\xi(k-\delta)$. The desired output $\mathbf{d}(k)$ in equation 5.3 for this network is a delayed version of the output $\theta(k-\delta)$ of the second network ('RCN2'). The later, has the current and the desired EE position as input. All the weights, including the output weights, are identical for both networks. The output of RCN2 are joint angles which are used to command the robot arm.

During the first time steps of the online learning process the model has no notion of how to change the joint angles to achieve the desired trajectory. Because of the random initialisation of the reservoir states in RCN2, this network will start to produce random joint angles. These are used by RCN1 to construct an inverse robot arm model by learning a relationship between the EE-positions over time and the commanded joint angles. Because of the weight sharing of the output weights, the controlling knowledge of RCN2 improves each iteration.

[2]depends on the value of the spectral radius. The connection weights are divided by the maximal absolute eigen value and multiplied by the spectral radius. Consequently, if the spectral radius is chosen smaller than one, thus operating in the non-chaotic domain, stability can be insured. However, if it is chosen larger than one the reservoir operates in the chaotic domain.

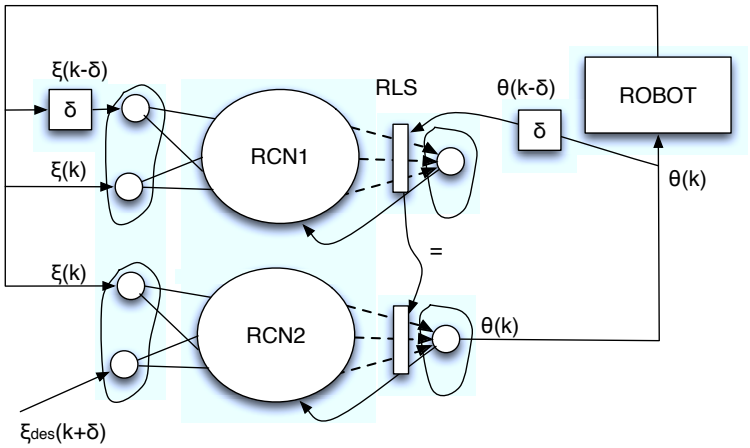


Fig. 5.1: Schematic representation of the proposed controller. The dashed arrows represent the output weights which are trained. These are the same for both networks. $\xi(k)$ is the end effector position and $\theta(k)$ the output vector with the joint angles.

parameter	description	value
N	reservoir size	$\in [300, 1000]$
ρ	spectral radius	1
λ	leak rate	1
ς	connection fraction RNN	$\in [10, 100]\%$
η	input scaling	0.001
κ	output feedback scaling	1/180
β	bias to reservoir	0.5
δ	time delay	1
α	learning rate	1

Tab. 5.1: Model parameters

This controlling approach learns to control the robot arm without any examples. However,mere example based training is possible by training the output weights of RCN1 offline. Afterwards, one can choose to continue optimizing this kinematic model online.

If example data is used to find an inverse kinematic model, the transient dynamics in the RC-network are used to achieve a similar reaching motion than the one seen during training. Even though, the desired EE position $\xi_{des}(k + \delta)$ was unseen.

The used model parameters are shown in Table 5.1.

5.3 Simulation

The results of some small experiments are presented. Figure 5.2 shows some reaching experiments with different converging times which demonstrates the spatiotemporal invariance. Each sample represents one iteration of the learning algorithm. Figure 5.3(left) illustrates a reaching motion after following a predefined trajectory.

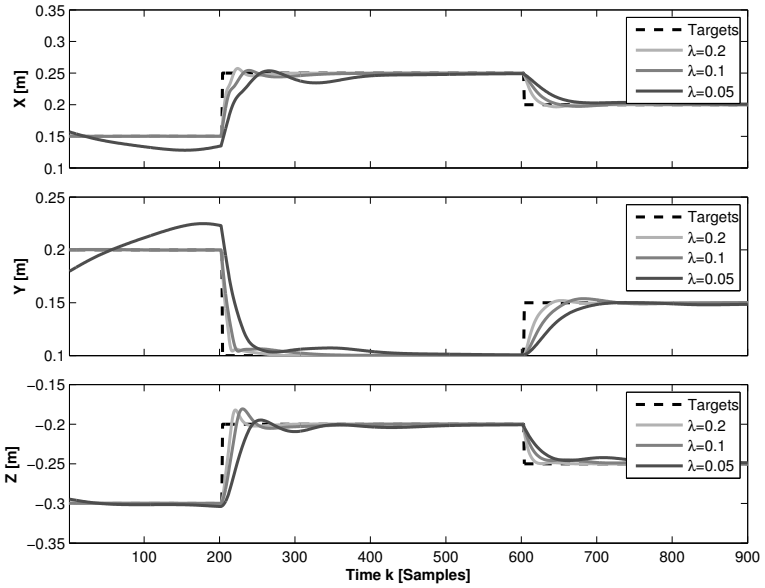


Fig. 5.2: Illustration of the simulation result during which the robot is commanded to reach different targets with different velocities. These plots show the X,Y and Z position of the end effector, respectively. The reaching velocity is regulated with the parameter λ which is used in equation 5.1. During this experiment the following parameters were used: a reservoir size of 400 neurons, $\delta = 1$, $\rho = 1$ (operating on the edge of stability), $\alpha = 1$ and input bias equal to 0.5

Figure 5.4 depicts a trajectory of the robots end effector during which the robot's feedback is perturbed. A discussion about these results will be given later in this chapter.

5.4 Functional Analysis

5.4.1 Dynamics and Nonlinearity

Each neuron features a $tanh()$ as nonlinearity. Because of these neurons and the interaction between them, the system dynamics are nonlinear. The used nonlinearity consists of a linear region around 0, so when the absolute input values are small, the dynamics are more linear. The chosen spectral radius determines if the RNN operates in a stable ($\rho < 1$) or chaotic regime ($\rho > 1$). Therefore, the nonlinear dynamics depend on the system input, the input scaling and the chosen spectral radius.

5.4.2 Attractor

The system's attractor type depends on its operation mode. When the learning algorithm (RLS) is applied continuously, the attractor is hard to determine. However, if the weight updates on a periodic or discrete signal are suspended after training, the attractor is respectively a limit cycle or point attractor.

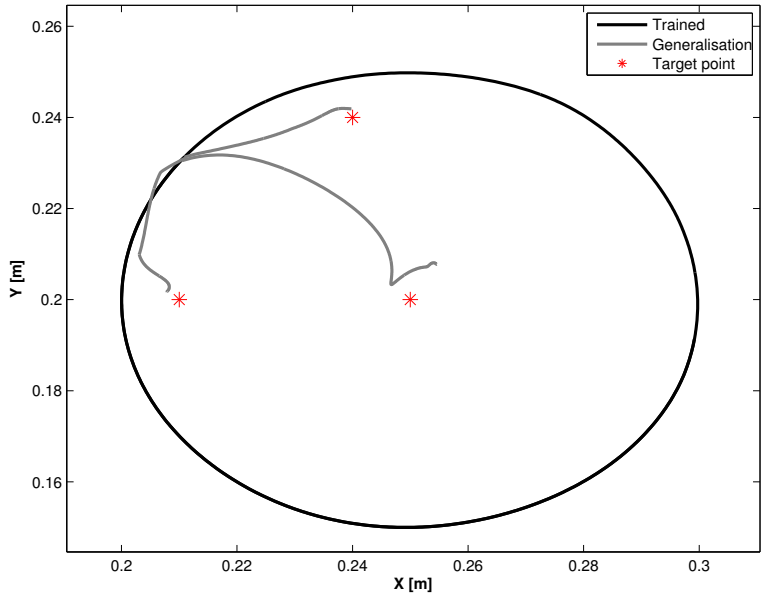


Fig. 5.3: This plot shows the results of a generalisation experiment in offline learning mode. The system is trained to move the robot's end effector in a circle. Afterwards, the training phase is stopped ($\Delta \mathbf{W} = \mathbf{0}$) and the robot is commanded to reach to an unseen target point. This experiment was repeated for different target points.

5.4.3 Coupling

The system generates an output vector containing joint angles for each DoF. Because these angles are generated by one dynamical system, the system is coupled.

5.4.4 Learning

Type The learning type of the system is supervised.

Algorithm The learning algorithm used is RLS but, with the used system strategy, other learning algorithms might be possible.

Mode The learning can be achieved in an online (continuous use of RLS) or offline (training phase after which RLS is suspended) manner.

5.4.5 Training Data

The proposed system does not need any training data. During operation, the systems starts generating joint angles that are used afterwards to construct a robot model. Each iteration, this model is improved by observing its controlling attempts. This means that the system will try to search a way to control the robot desirably.

However, the internal system model can be trained with example data as well. An example data set containing a trajectory and joint angles, can be used to train the system in offline training mode. However, to achieve a good robot representation, a large amount of training data is recommended.

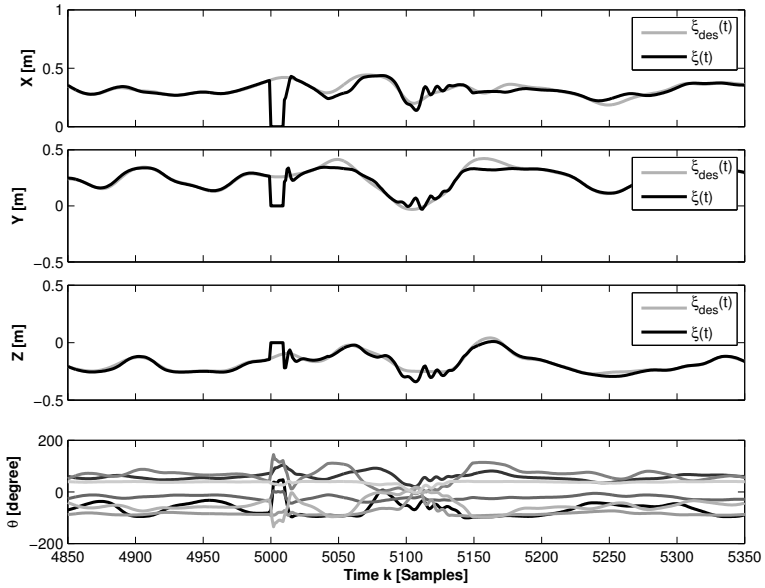


Fig. 5.4: Illustration of an experiment in online training mode, during which the robots trajectory if perturbed. The upper tree plots show the X,Y and Z end effector position. The bottom plot shows the commanded joint angles (7 joints) during this perturbation. The introduced perturbation occur during 10 iterations. Besides a leak rate of $\lambda = 1$, the same system parameters were used as in the previous experiment.

5.4.6 Generalization

The generalisation capabilities should be discussed separately for each learning mode:

- **online**: When the learning algorithm is used throughout the whole task, the generalization capabilities are attained by the learning algorithm itself. This operation mode allows to model changes in the environment or internal robot structure.
- **offline**: When the learning is suspended after the training phase, the generalization abilities are acquired by the transients of the RC-networks. In Figure 5.3(left) the reaching motion of unseen target points is shown together with the training trajectory.

If the generalization capability is insufficient, the described NDMPG (Neural Dynamical Motion Primitives Generator, Chapter 4) or other trajectory generating modules can be used to insure generalization in the sense of generating similar trajectories as seen during training (Figure 5.3, right). In this case, the NDMPG is attached to the $\xi_{\mathbf{des}}(t + \delta)$ system input.

5.4.7 Modulation

The described systems reaching speed can be modulated by changing the leak rate λ. This is demonstrated for different λ's in Figure 5.2. The leak rate should only be changed during the testing phase when using the offline learning mode. However, when using the online learning mode, the reaching velocity can be modulated at any time. The desired end effector position is modulated by $\xi_{\mathbf{des}}(k)$. The transition between

different reaching points can modulate the shape of the actual reaching motion. In fact, this transition can regulate the velocity as well.

5.4.8 Sensory Feedback Integration

The proposed system is build to integrate sensory feedback. In the example of the iCub robot arm this means that after commanding the robot with joint angles, the encoder values are measured after a given time step. These values can be used to calculate the actual end effector position, and is afterwards used as system input. Other relevant sensor information, which can help to improve the internal robot representation, can be used as well. These would be added to the system's input vector.

5.4.9 State Variables

The amount of state variables is equal to the amount of neurons used in the system and is uncorrelated with the DoF.

5.4.10 Robustness and Adaptation to Perturbations

As demonstrated in Figure 5.4, the proposed system is able to handle unforeseen perturbations. In this example the robot's end effector is forced to an undesired position for 10 time steps. Afterwards the system converges again to the desired trajectory. Because of operating in the online learning mode, the system tries to change its internal model according to the perturbation. The effect on this model can be noticed when the perturbation is removed (between samples 5100 and 5150).

The error in the learning algorithm is defined on the accuracy of the internal model and not on the end effector position. Reaching the end effector position is the result of a good internal model. This strategy introduces the possibility to handle changes in the robots pose, construction, disabilities and/or environment. It should be noted that the transition between different models can lead to fast changing transient behaviour because of the fast learning nature of RLS.

5.4.11 Stability

The proposed system uses a RC-network which has a given input. The neuron states are bounded to $[-1, 1]$ because of the used nonlinearity. The output is a linear mapping of these neuron states. This means that the system output is bounded as well. Although in most cases the desired attractor is reached, convergence can not be proven.

5.5 Non-Functional Analysis

5.5.1 Representation and Interface

At this point the described system generates joint angles in degrees. These generated values are filtered, limiting them to the possible angular values. The end effector position is calculated in a Cartesian coordinate system and used as system input. The end effector position needs to be scaled down to meters to insure no neuron state saturation. Also the generated angles are scaled down, by dividing them by 180, before given to the reservoir as output feedback. As stated before the system has different learning modes. The online learning mode is able to adapt to radical changes in the robot or its environment. The offline learning mode, on the other hand, is unable to adapt to such changes. In Table 5.2 we give an overview of the interface parameters.

parameter	description	value
N	reservoir size	$\in [300, 1000] \sim$ [faster, precision]
$\xi(k)$	robot feedback	X,Y and Z in (mm)
$\xi_{\mathbf{des}}(k)$	desired position	X,Y and Z in (mm)
$\theta(k)$	system output	degree $(^\circ)$
λ	leak rate	$\in [10^{-6}, 1]$
δ	time delay	1

Tab. 5.2: Interface parameters

5.5.2 Timing

Depending on the learning mode, the used system strategy has an upper time limit[3]. During online learning the equations 5.5 and 5.6 need to be calculated. Each iteration takes at least the time needed to calculate these equations. This calculation depends on the reservoir states and can change during operation. When the offline learning mode, the weight updating is stopped during the testing phase. As a result, those equations do not need to be calculated afterwards.

5.5.3 Robustness and Reliability

When using the online learning mode without examples, it is possible that the learned motion limits the robot to achieve a desired trajectory perfectly. Therefore, although quite robust, converging anytime can not be guaranteed. This also applies for the offline learning mode. Because of the limited calculations, a good Real-time constraint is possible.

5.5.4 Dependencies

The system depends now only on proprioception and determines the end effector position by calculating the forward kinematics from the encoder values. It would be possible to only depend on exterioception to determine the end effector position. Furthermore, the system is implemented in MatLab and is communicating with the robot over the YARP robot platform.

5.5.5 Runtime

As indicated before the system provides online and offline learning. When using online learning, equation 5.1, 5.2, 5.5 and 5.6 need to be calculated each iteration. However, when using offline learning, only equations 5.1 and 5.2 need to be computed during testing ($\Delta\mathbf{W} = \mathbf{0}$).

5.5.6 Usefulness for Recognition

The control of fast dynamical systems need a rather small time delay δ and a larger leak rate λ. When observing $\Delta\mathbf{W}$ during the online learning mode, one can identify a sudden change in the environment or the robot itself when $\Delta\mathbf{W}$ becomes significantly larger. After adjusting the internal system model these weights will get smaller again. This shows that the system can detect spatiotemporal changes in its input.

[3]On a Quad Core system with 8GB of RAM, for a reservoir with 500 neurons the calculations in the online mode take around 0.05 seconds. In offline mode this is around 0.005 seconds.

5.6 Summary

In this chapter, a system called Neural Motion Primitive Control is proposed which uses a recurrent neural network (RNN) to model the dynamics of a robot. The training is done by using an RLS based training technique and can operate online or offline. By using the transients in this RNN, similar motion as seen during training can be generated. The robustness, modulation and generalisation capabilities are demonstrated with some simple experiments. This system should be used in a combination with a trajectory generator like NDMPG to achieve similar trajectories as seen during training.

Chapter 6

Neural Dynamic Movement Primitives

Andre Lemme, R. Felix Reinhart, Matthias Rolf, Jochen J. Steil
UniBi

6.1 Short Introduction

Neural dynamic movement primitives (NDMP[1]) are a combination of the ideas from reservoir computing and associative learning, which results in an elegant formulation of neural dynamics. NDMP couple task and joint space by means of an attractor-based reservoir network (see Fig. 6.1). The combination of unsupervised and supervised learning methods allows to generate movements with biologically plausible velocity profiles and to solve the inverse kinematics at the same time.

The main operation modes of this model are:

1. Association:
   In this mode, the model is used attractor-based: A target position $\mathbf{u}^*$ is clamped to the network input while the network is iterated until the dynamics converge. Then, the resulting joint values $\mathbf{q}^*$ are read out. In this way, it is possible to compute the inverse/forward kinematics of the robot.
2. Autonomous reaching by feedforward-feedback control:
   This is the main operation mode to generate dynamic motion. The basic idea is to exploit the network's transient dynamics while approaching an attractor state for smooth and robust movement generation.

6.2 Model Description

The NDMP framework uses the associative neural reservoir learning (ANRL) approach, which is described in the following. The associative nature of the neural model allows for learning of forward and inverse kinematics in parallel and in a single network. The mapping is bi-directional: we can use task coordinates as input and joint space coordinates as output or vice versa. These two mappings functionally implement the forward and backward models described in [24] and use afferent motor copies for sensory prediction of the input.

[1] The case discussed here is the system introduced in [23].

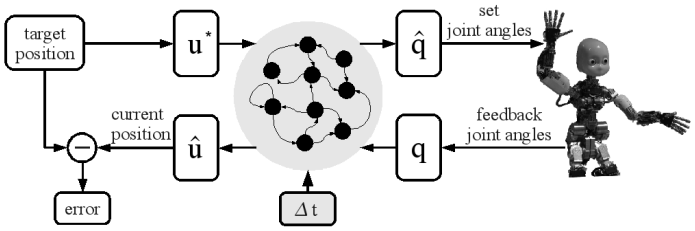


Fig. 6.1: The associative reservoir network architecture connects task- and joint-space variables bidirectionally. Thereby the inverse and forward kinematic models can be be queried continuously for generalization.

Formally, the architecture comprises a recurrent network of nonlinear neurons that interconnect inputs $\mathbf{u}$ and outputs $\mathbf{q}$ shown in Fig. 6.2a. We denote the network state at time step k by $\mathbf{z}(k) = (\mathbf{u}(k)^T,\ \mathbf{y}(k)^T,\ \mathbf{q}(k)^T)^T$, where $\mathbf{u}, \mathbf{y}$ and $\mathbf{q}$ are the input, reservoir and output neurons respectively. W^{net} captures all connection submatrices between neurons in the network and is defined by

$$\mathsf{W}^{net} = \begin{pmatrix} 0 & \mathsf{W}_{res}^{inp} & 0 \\ \mathsf{W}_{inp}^{res} & \mathsf{W}_{res}^{res} & \mathsf{W}_{out}^{res} \\ 0 & \mathsf{W}_{res}^{out} & 0 \end{pmatrix}, \tag{6.1}$$

where we denote by $\mathsf{W}_{\star}^{\square}$ all connections from $\star$ to $\square$ using *inp* for input, *out* for output, and *res* for inner reservoir neurons. Only connections W_{res}^{out} and W_{res}^{inp} projecting to the input and output neurons are trained by error correction (illustrated by dashed arrows in Fig. 6.2a). All other weights are initialized randomly with small weights and remain fixed. We consider recurrent network dynamics

$$\mathbf{x}(k+1) = (1-\Delta t)\,\mathbf{x}(k) + \Delta t\,\mathsf{W}^{net}\mathbf{z}(k) \tag{6.2}$$

$$\mathbf{z}(k) = \sigma(\mathbf{x}(k)), \tag{6.3}$$

where for small Δt continuous time dynamics are approximated. $\mathbf{z}$ is obtained by applying activation functions component-wise to the neural activations $x_i, i = 1 \ldots N$. We use parametrized logistic activation functions $y_i = \sigma_i(x_i, a_i, b_i) = (1+\exp(-a_i x_i - b_i))^{-1}$ for the reservoir neurons. Input and output neurons have the identity as activation function, i.e. are linear neurons.

6.3 Simulation

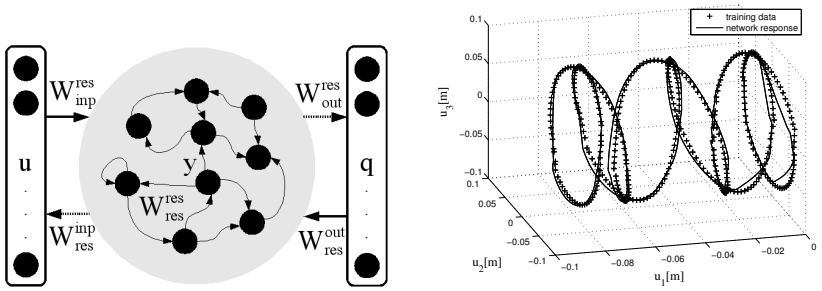
In this simulation, we show the performance of the NDMP model in a reaching movement scenario for the right arm of iCub. We use a spiral-like motion as training pattern. In task space, end effector positions are defined by

$$u_1(k) = -0.08\ \ (0.5\,sin(\bar{\omega}\,k) + 0.5) \tag{6.4}$$

$$u_2(k) = 0.08\,cos(6\,\bar{\omega}\,k) \tag{6.5}$$

$$u_3(k) = 0.08\,sin(6\,\bar{\omega}\,k). \tag{6.6}$$

This motion is constrained to a cylinder of $8cm$ in length and with a diameter of $16cm$ (see Fig. 6.2b). We set $\bar{\omega} = 2\pi/400$ and record two pattern periods ($K = 800$ samples) as training data $(\mathbf{u}(k)^T, \mathbf{q}(k)^T)^T$ using a Jacobian-based inverse kinematics



(a) (b)

Fig. 6.2: The reservoir network architecture (a) and the training data in task space with network response (b).

solver. The Cartesian end effector coordinates (u_1, u_2, u_3) are presented in meters and have their point of origin at the end effector position of iCub's right arm in the home position. To excite the network with values in a reasonable range, the network gets task space inputs in decimeters and joint angles in radians. The network parameters, which are used in this simulation are shown in Tab. 6.1.

We calculate the positioning error for the iCub arm-movements in task space by

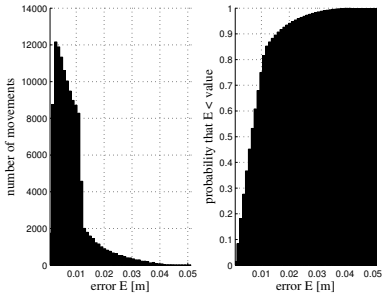
$$E(u_1, u_2, u_3) = ||(u_1, u_2, u_3) - FK(\hat{IK}(u_1, u_2, u_3)))||,$$

where FK denotes the known analytic forward kinematics and $\hat{IK}$ the learned inverse kinematics. We compute errors in task space since errors in joint space can be misleading: the network could find a different solution of the redundant kinematics than the analytic model in order to approach the same target position.

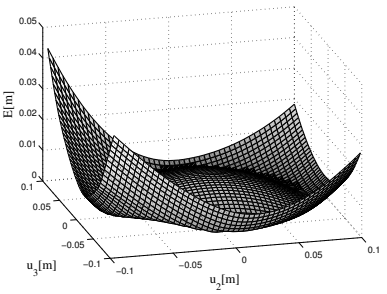
The iCub arm kinematics are trained on the spiral data shown in Fig. 6.2b, where the positions corresponding to the output of the trained network are superimposed. To test the network's generalization of the inverse kinematic mapping systematically, we query network outputs for targets sampled from a three dimensional and equally spaced grid with 50x50x50 vertices that spans a cuboid of 20x20x10cm in task space. The error histogram and cumulative distribution for all 125.000 targets in the cuboid are shown in Fig. 6.3a. 75% of the 125.000 target positions are reached with an residual error less than 1cm. The maximal error is below 5cm.

	iCub	
Reservoir Size	200	
Connection	ρ	a
Input-Reservoir	0.4	0.4
Reservoir-Reservoir	0.2	0.05
Reservoir-Output	0.0	0.0
Output-Reservoir	0.4	0.1
Input-Output	–	–
Learning	η	
IP	0.00002	
BPDC	0.01	
Cycles	1000	

Tab. 6.1: Parameters for network construction and learning.

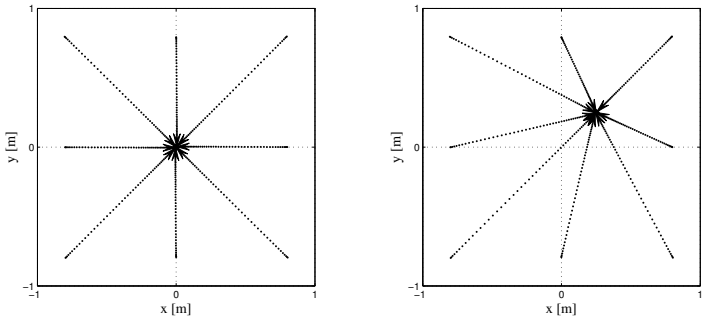


(a) Error histogram and cumulative distribution in the cuboid.



(b) Reaching errors in the cuboid projected onto the frontal plane.

Fig. 6.3: Performance error of the learned inverse kinematics in a generalization task on iCub.



(a) (b)

Fig. 6.4: Exemplary movements from several initial positions to two different targets: $\mathbf{u}^* = (0, 0)$ in (a) and $\mathbf{u}^* = (0.25, 0.25)$ in (b).

In order to visualize the error distribution spatially in Fig. 6.3b, we project errors in the cuboid onto the frontal plane by marginalization with respect to the u_1 axis. The error surface is smooth, which means that similar errors can be expected for nearby targets. Additionally, we have a graceful degradation of performance. The reservoir network achieves a mean precision of $0.8cm$ in the cuboid and generalizes well to targets away from the training examples, although the sparsely sampled 800 training samples cover only a small subset of the three dimensional volume (compare Fig. 6.2b).

Fig. 6.4 illustrates the movement generation capabilities of the the attractor-based network: When the network dynamics are iterated with a fixed target position at the input, straight reaching trajectories are generated. Note that the movement speed depends on the relative distance between current position and the target (compare step widths between dots in Fig. 6.4 (left) and (right)).

6.4 Functional Analysis

6.4.1 Dynamics and Nonlinearity

The dynamics of the system are given by the reservoir network update equation (6.2). The nonlinearity is constituted by the parametrized sigmoidal functions $\sigma_i(\cdot)$ of the reservoir neurons, and the recurrent network dynamics. The interaction between synaptic- and intrinsic plasticity are the crucial factors in the dynamics of the system.

6.4.2 Attractor

The network represents a smooth mapping by a continuum of fixed-point attractor states. A specific association of task and joint variables is queried by driving the network with a desired input $\mathbf{u}^*$. The network then converges to the related attractor state and provides the desired outputs $\hat{\mathbf{q}}$ (compare Fig. 6.1).

6.4.3 Coupling

Multiple DOFs are controlled by a single network using a distributed representation of the entire robot configuration in the state space of the recurrent network, i.e. $\mathbf{z}(k) = f(\mathbf{u}, \mathbf{q}, \mathbf{z}(k-1))$.

6.4.4 Learning

We start by training the dynamic network with trajectories, e.g. demonstrated by a teacher, to learn the kinematic mapping. This training makes no explicit reference to the form of the trajectory, which consequently is not stored directly in the network. Learning proceeds supervised, i.e. needs corresponding task/joint space data pairs for training, i.e. $\{(\mathbf{u}(k), \mathbf{q}(k)\}_{k=1,\ldots,K}$. In principle, it is possible to train the associative reservoir model with different algorithms in online or offline training modes. We typically apply the following combination of online learning techniques for intrinsic and synaptic plasticity:

Synaptic plasticity:

Type Supervised read-out learning

Algorithm Backpropagation-Decorrelation (BPDC) learning [25]

Mode Online learning.

Intrinsic plasticity:

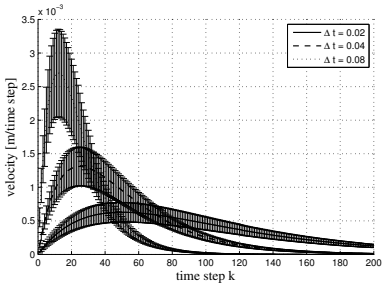
Type Unsupervised reservoir optimization

Algorithm Intrinsic plasticity (IP) [26]

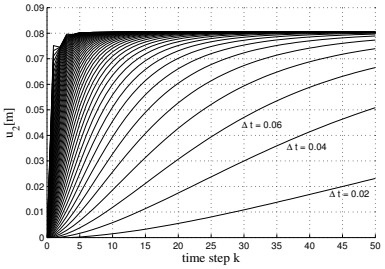
Mode Online learning.

6.4.5 Training Data

NDMP learns to map end effector position to joint values from training data that comprises end effector trajectories together with the corresponding trajectories in joint space. Although the performance of the network will increase with more training samples, it was shown in [27] that a training set with only 250 data points is sufficient to learn the inverse kinematics for a 7DOF robot arm in a reasonable area. In this scenario, training data was even aquired by recording pairs of end effector coordinates and joint angles teached by a human tutor. However, any expert system can be used



(a) Averaged speed profiles with standard deviations.



(b) Step response of the network.

Fig. 6.5: Dynamic properties of generated reaching movements for different gains Δt.

for data acquisition as well. It is then important to distribute the training data such that the samples cover the area of operation.

6.4.6 Generalization

The NDMP framework has excellent generalization abilities due to the distributed and non-local representation of inputs in the reservoir state space. We consider spatial generalization of the inverse kinematic mapping in the generalization tests: New targets are set at the input of the network and the transient trajectory toward the target attractor state is observed. Given that all movement directions needed for the task are present in the training data, the performance of the generalization scales to complex joint configurations [28].

Another important property of the kinematic learning is that constraints are preserved and can be generalized: It is shown in [27, 29] that constraints in joint space present in the training data are learned in the network and are also generalized by means of spatial variation of the target position.

We apply a goal as input, however, as we couple both spaces, we could also set the goal in joint space. Moreover, incomplete or combined goals in both task and joint space can be considered. The missing components can then be completed by convergence to the associated attractor state.

6.4.7 Modulation

In the NDMP framework, there are several ways of modulation on different levels possible. Firstly, the speed of generated movements can be modulated by the scalar parameter $0 \leq \Delta \leq 1$ in (6.2). Δt can be understood as step width of the network update equation (6.2). The step size allows to modulate the velocity profile and duration until convergence (see Fig. 6.5a and Fig. 6.5b).

Secondly, the mapping learned by NDMP can be modulated by additional inputs. In [30], Rolf et al. showed that it is possible to learn a parameterized inverse kinematic model of a humanoid robot for flexibile tool use. This can be extended to arbitrary complex mappings in principle, if enough training data is provided.

6.4.8 Sensory Feedback Integration

One can extend the number of controlling DOFs or sensory feedback by clamping the variables to the input/output neurons of the reservoir (see Fig. 6.1). This possibility

allows the user to adapt the learning to various problems. For instance, one could add the torque sensor information of the robot to the input in order to detect movements with external forces applied to the robot.

6.4.9 State Variables

Besides of the input/output variables, the system has a high-dimensional state representation which holds the current activation of the reservoir. Inputs are mapped to reservoir states in a distributed manner. Learning mechanisms such as intrinsic plasticity further lead to a more sparse but still non-local encoding of inputs [31]. In addition, the reservoir encodes the temporal history of inputs by means of transient network states.

6.4.10 Robustness and Adaptation to Perturbations

In [32] it was shown that the network can adapt to perturbations in the reaching process including spatial perturbations, i.e. spatial generalization of the kinematic mapping, and change in the posture positions during reaching. The feature of NDMP is that it can approach the target from arbitrary directions, which makes the approach robust against perturbations. Therefore, also switching goal positions is possible while the reaching movement is performed.

6.4.11 Stability

Depending on the specific network configuration, global input-stability can be assured by initialization. This applies to all echo state architectures without output feedback connections W_{out}^{res}. Learning in the system further complicates the stability analysis. The intrinsic plasticity mechanism often used to improve the reservoir encoding capabilities can drive the reservoir dynamics towards instability.

There is a mechanism for online stabilization in case of output feedback [33], however, the criterion this mechanism relies on is very conservative. Experiments show that the entire NDMP system is locally stable. Even more strict: If the NDMP system is not stable, NDMP cannot generalize a smooth mapping and do not generate smooth movements. Therefore, it is not necessary to monitor stability in each step during the training. To assure safe operation on the robot, a evaluation step can be performed after learning.

6.5 Non-Functional Analysis

6.5.1 Representation and Interface

The input/output data depends on the task. Typically, we configure the NDMP network to map end effector positions to joint values. The end effector positions can be defined in a local or a global coordinate system and should be scaled to a reasonable range e.g [-1,1]. A possibility is to use the units meters for end effector positions and radians [rad] for joint values. If the values are in degree, the absolute inflow into the network can cause the activation functions to saturate and thus disturb learning of the smooth mapping.

The main parameters and their default values are shown in Tab. 6.1. We divide the parameter set into two subsets: The first parameter set controls the architecture of the model, e.g. the size of the reservoir and its initial weights (Tab. 6.1 Connection & Reservoir Size). The other set of parameters determines the learning duration and the size of each learning step (Tab. 6.1 Learning).

6.5.2 Timing

The NDMP model has an internal timing parameter (Δt in (6.2)) which can scale the network update step continuously. Δt therefore determines also the speed of generated movements (compare Sec. 6.4.7). This parameter is set to unity during training and needs to be adapted depending on the robot's physcial dynamics for movement generation.

6.5.3 Robustness and Reliability

The NDMP framework can give an anytime guarantee and also fulfill real-time constraints, if the framework is in evaluation mode, because the computational complexity is constant (see Sec. 6.5.5).

Regarding the network parameters, we draw on the well known robustness properties of reservoir networks, which have been shown to work for a large variety of connectivity schemes and output learning methods [34]. Learning of the associative reservoir is very robust to changes of the parameters in the introduced ranges (see Tab. 6.1). We do not observe any overfitting and therefore do not need to determine an optimal number of neurons and connections very carefully. In particular, the intrinsic plasticity rule has a strong input specific regularizing effect: it keeps neurons in a reasonable working range for their specific relevant input [26, 31]. In addition, intrinsic plasticity reduces the dependency on the initialization of the fixed portion of the network parameters and limits the respective variance, as has been shown in [31].

6.5.4 Dependencies

NDMP is a neural learning scheme that is completely data driven which causes an intricate dependence on the training data: The structure of the training data is important and determines the network performance in several ways. Besides the importance to distribute the training data such that the samples cover the area of operation, there is a trade-off between the desired coupling of the DOFs and generalization: constraints that are systematically present in the data will be learned by the network and reproduced. For example, if the virtual drawing on a board is trained, then generalization is possible only within the plane of this board [28], or if handling of a stick with both hands is trained, then it is not possible to move the hands independently [29]. In summary, to enable generalization, respective movement directions need to be present in the training data.

6.5.5 Runtime

NDMP has four different modes:

1. Learning:
   (a) offline learning
   (b) online learning
2. Evaluation
   (a) transient-based movement generation
   (b) association

In the offline learning mode (1a) a set of data samples is processed, which needs some time. If N is the amount of neurons in the network, then the computational complexity of this learning mode is in the $O(N^3)$ complexety range. Unlike offline learning, each step of the online learning (1b) has much less computational cost ($O(N)$) and can be done in parallel to the evaluation mode. The evaluation has two different modes

as mentioned in Sec. 6.1. The first evaluation mode (2a) covers the transient-based generation of movements. In this mode, the network state has to be update according to (6.2) in order to compute the next output values. For one single evaluation step of the network, the complexity is $O(N^2)$ in general. However, using a sparse matrix implementation and a sparse reservoir initialization (which is typical), the cost for the network update can be reduced to $O(N)$. If we evaluate in association mode (2b), the network dynamics are iterated several times until the network is converged. Convergence can take several iteration steps depending on the specific network architecture and reservoir initialization [32].

6.5.6 Usefulness for Recognition

The ARNL has a fixed random weight matrix to transform the input into the state space. However, the IP learning rule form the state depending of synaptic summation in each neuron in the reservoir. That means that you will find similar states for similar inputs for the same random weight matrix this feature can be used for recognition.

6.6 Summary

The neural dynamic movement primitives (NDMP) framework is a computational model that combines learning of a forward and inverse model with movement generation in an associative neural reservoir learning scheme (ANRL). NDMP learn the kinematics fully data-driven and through coupling of all inputs and joints in a single network. Movements are generated by retracing the transients to network attractor states. Spatial generalization is performed by clamping network inputs to target values which modulate the internal attractor state. While NDMP without extensions do not learn specific trajectory shapes, the framework generates straight reaching movements with generic, bell-shaped velocity profiles while solving the inverse kinematics at the same time. Therefore, NDMP provides a building block that links task with joint space and may combine towards richer motor skills for smooth and natural movements.

Chapter 7

Planned Motion Primitives using Approximate Inference

Elmar Rückert, Gerhard Neumann
TUG

7.1 Short Introduction

Many complex behaviors can be decomposed into simpler movements, however, how to do this decomposition is unclear. One intuitive approach is to represent a movement as a sequence of subgoals (also called via-points) $\mathbf{g}_i$ which the agent has to reach and a corresponding timing parameter d_i indicating the time to reach the subgoal. We will denote this sequence of subgoals as movement primitive.

We assume that we can employ a planning machinery which can guide the agent (at least locally) to any given subgoal in a given amount of time. The advantage of the subgoal representation is that it is very compact, i.e. the number of parameters to describe a motion is small compared to other motion primitive approaches. We want to address the question whether such an compact representation of a movement can facilitate learning at the level of the motion primitive parameters.

For planning, we will use a state-of-the-art method based on probabilistic inference, called Approximate Inference Control (AICO, [35]). AICO is used to solve stochastic optimal control (SOC) problems. Many movements measured from animals or humans share various characteristics (like the resulting bell-shaped velocity profile) with the SOC solution, indicating that animals also solve a SOC problem.
AICO is a local planner, i.e. given an initial solution and a cost function, AICO finds a locally optimal solution which minimizes the given cost function. The parameters of the motion primitive (the subgoals) directly modulate the cost-function used for AICO. The algorithm is used to plan the movement to the next subgoal $\mathbf{g}_i$ in a fixed amount of time (d_i). After d_i seconds, when the agent is supposed to reach the current sub-goal $\mathbf{g}_i$, AICO is used to acquire a new plan in order to reach $\mathbf{g}_{i+1}$.

AICO is (like many other planning methods in continuous spaces like [36, 37]) a local optimization method, thus, for complex movements we already have to provide an trajectory which is close to the optimal one. However, we will use AICO only to reach sub-goals in a neighborhood of the current state. Thus, a single sub-goal is

typically easily reachable. In this case the planner can be called with trivial initial solutions like using no control at all.

The AICO algorithm assumes full knowledge of the system dynamics. In our experiments we will use the same assumption, however, learning the system dynamics is part of our future work and has already been done in [38,39] using Locally Weighted Projection Regression as function approximator. Thus, when using a (local) planning machinery like AICO, learning takes place at two stages, i.e. learning the system dynamics (supervised learning) and learning appropriate subgoals to fullfill a given task (reinforcement learning).

7.2 Model Description

A movement primitive is represented by a sequence of subgoals $\mathbf{g}_i$ and a corresponding timing parameter d_i. There are different ways to parametrize a single subgoal $\mathbf{g}_i$:

- **Positions:** A subgoal $\mathbf{g}_i$ is given by the desired joint-position θ_i and the time to reach this position. All desired joint-velocities for the subgoal are set to 0, thus, the subgoal typically represents a tuning point in the joint trajectory. In addition, we use an individual parameter k_i to balance the costs between reaching the subgoal and the needed energy of the movement. In this representation the subgoal has $n + 2$ number of parameters, where n is the number of joints of the robot.
- **Position and velocities:** In addition to the joint positions, we can also define the desired joint-velocities $\dot{\theta}_i$ of the agent when reaching the subgoal. A subgoal has now has $2n + 2$ parameters.
- **Importance factors:** Furthermore, we may define importance factors $\mathbf{a}_i$ for each dimension of θ_i and $\dot{\theta}_i$ indicating the importance for the planner to reach a certain position (or velocity) with joint j in relation to the subgoals of all other joints.

These representations have an ascending number of parameters, however, the expressibility of the subgoals is also increased. We still have to evaluate which of these representations should be preferred. Our current experiments only cover position-based subgoals. Another promising extension of this approach is to use subgoals not in joint space, but in task space such as the end-effector position.

In order to guide the agent to the desired subgoal, we use the Approximate Inference Control (AICO) algorithm to solve a SOC problem. The cost function of the SOC problem is chosen such that the agent navigates (if possible) to the given subgoal in the given amount of time. The cost function c_t for intermediate time steps ($t < d_i$) is just given by the used energy, i.e.

$$c_t(\mathbf{x}_t, \mathbf{u}_t) = -k_u \mathbf{u}_t^T \mathbf{u_t}$$

The final costs at the final time step $T = d_i$ are given by the squared distance to the sub-goal

$$c_T(\mathbf{x}_T) = -(\mathbf{x}_T - \mathbf{g}_i)^T \mathrm{diag}(\mathbf{a}_i)(\mathbf{x}_T - \mathbf{g}_i),$$

where the elements of $\mathbf{a}_i$ are set to 1 for joint positions and to 0.01 for joint velocities. AICO reformulates the stochastic optimal control problem as an Bayesian inference problem. This alternative view of control and planning has many advantages:

- Probabilistic Inference might help to bring different disciplines (planning, AI, imitation learning ...) in robotics together.
- Structured representations like graphical models can be used

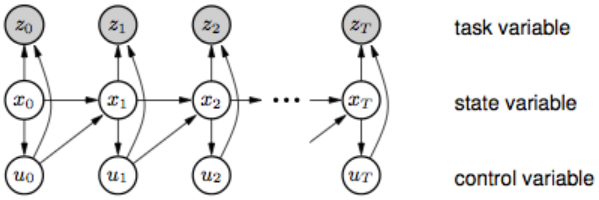


Fig. 7.1: Graphical Model for Probabilistic Planning from M. Toussaint [40]: A stochastic optimal control problem can be formulated as probabilistic inference problem.

- Many algorithms exist to exploit this structure (Massage-Passing, Loopy Belief-propagation, Variational Inference, Extended Kalman Filters, Particle Filters ...)

The graphical model representation for a SOC problem is illustrated in Figure 7.1. The state variable $\mathbf{x}_t$ denotes the joint angles and joint velocities. Controls are labelled by $\mathbf{u}_t$. The time horizon is fixed to T time steps.

The task variable z_t expresses a performance criteria (like avoiding a collision, or reaching a goal). It is given by:

$$P(z_t = 1|\mathbf{x}_t, \mathbf{u}_t) = \exp(-c_t(\mathbf{x}_t, \mathbf{u}_t)), \tag{7.1}$$

where the function c_t represents the costs in classical stochastic optimal control. The SOC problem is now solved by calculating the posterior $P(\mathbf{x}_{1:T}, \mathbf{u}_{1:T}|z_{1:T} = 1)$ over the trajectories, conditioned on observing a reward ($z_t = 1$) at each time step t. It can be shown that (under some assumptions...) this is equivalent to computing the classical SOC solution [40]. The posterior can be computed by using message passing in the given graphical model. The belief $b(\mathbf{x}_t)$ of a state is given by:

$$b(\mathbf{x}_t) = \mu_{\mathbf{x}_t \to \mathbf{x}_{t+1}}(\mathbf{x}_t)\mu_{\mathbf{x}_{t-1} \to \mathbf{x}_t}(\mathbf{x}_t)\mu_{\mathbf{z}_t \to \mathbf{x}_t}(\mathbf{x}_t), \tag{7.2}$$

where $\mu_{\mathbf{x}_t \to \mathbf{x}_{t+1}}(\mathbf{x}_t)$ denotes the forward message, $\mu_{\mathbf{x}_{t-1} \to \mathbf{x}_t}(\mathbf{x}_t)$ the backward message, and $\mu_{\mathbf{z}_t \to \mathbf{x}_t}(\mathbf{x}_t)$ is expressing the message of the immediate task variable.

Exact massage passing can only be derived for the special case of linear dynamics, quadratic costs, and Gaussian noise (LQG). In this case all messages are Gaussians and can be computed in closed form, see [40].
For non-linear dynamics we can use a wide range of approximate inference methods (extended Kalman smoothing, Unscented Transform (UCT), or particle methods suggested by Hoffman et. al. [41]).
The simplest method is extended Kalman smoothing, which is also used in our experiments. The non-linear system is linearized at the current mode of the belief $b(\mathbf{x}_t)$. At this point of linearization again simple Gaussian messages are used, as in the LQG case. Subsequently, the system is linearized at the new mode of the belief. The algorithm loops back-and-forth over t and updates local messages until the belief of the trajectory converges.
The algorithm is similar to the iterative Linear Quadratic Gaussian (iLQG) planner proposed in [42] and [36]. The only difference is that forward messages are neglected in the iLQG algorithm, which leads to slower convergence.

The resulting optimal policy in AICO is given as a closed-loop non-linear feedback controller of the form:

$$\mathbf{u}_t \quad = \quad \mathbf{l}_t + L_t\mathbf{x}_t, \tag{7.3}$$

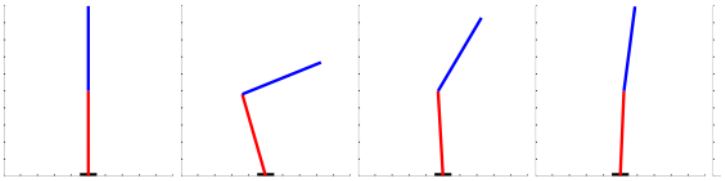


Fig. 7.2: Atkeson et. al. [43]: Illustration of a complex 2-link balancing behaviours for a 2 link balancing task. The figure shows the response to a 25 Ns push.The optimal strategy is to perform a fast bending movement and subsequently return to the upright position.

where $\mathbf{l}_t$ is a constant control term and L_t represents the linear control gain matrix. Both quantities vary for each time step, resulting in a non-linear control law.

7.3 Simulation

We conducted preliminary experiments on a complex 2-link balancing task (see Figure 7.2). The physical properties of the robot were chosen to match a human. For more details, see [43]. The robot gets pushed by an external force and has to keep balance. The optimal strategy is to perform a fast bending movement and subsequently return to the upright position. This is a highly non-linear control policy.
The trajectory was modelled with two subgoals resulting in two sub-trajectories. The first trajectory is optimized with respect to the first subgoal. The second trajectory stabilizes the agent in an upright position. The optimal subgoals were learned using the CMA-ES algorithm [44], which is a state-of-the-art algorithm for learning motor skills. Our learning strategy with multiple subgoals is able learn a near optimal (there is still some work to do) policy within 15 minutes of simulation time or 200 trials. See Figure 7.3 for the resulting trajectories. Using only a single subgoal fails because of the highly non-linear movement.

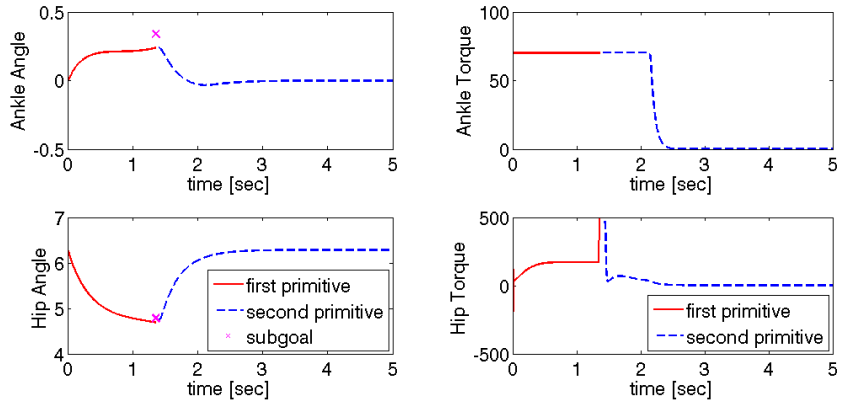


Fig. 7.3: Results for the 2-Link Balancing Task: The robot needs to keep balance after a push of 25 Ns. Figure (left) illustrates the near optimal hip and ankle angle trajectories. The subgoal is indicated by the magenta marker and could not exactly be reached because of torque constraints (right).

7.4 Functional Analysis

7.4.1 Dynamics and Nonlinearity

The dynamics of the system has to be known. The AICO algorithm deals with non-linearities by linearizing the system at the current mode of the belief of the trajectory. The motion primitive itself has no dynamics of its own, except for the executing time of the primitive. The execution time is needed for determining the time which is left to reach a subgoal.

7.4.2 Attractor

The AICO approach returns a non-linear feedback controller to follow the mode of the posterior distribution over all trajectories. Thus, the posterior is also an attractor of the system, at least in a local neighborhood of this trajectory.

7.4.3 Coupling

The coupling between joints is inherent to the approach because the known system dynamics are used. Coupling with sensory variables is achieved by the feedback controller, however, how to influence the subgoals itself by sensory input is unclear.

7.4.4 Learning

Learning takes place at two levels. At the lower level we have to acquire the system dynamics, which is a standard supervised learning problem. Many different approaches can be used for this setup (see [45] for a comparison of some of the most popular model learning algorithms). At the level of the subgoal parameters we have to employ reinforcement learning, we are currently using a state-of-the art method for motor skill acquisition called CMA-ES ([44]). On the long run we also want to employ an approximate inference planner using learned models at this level.
So far we have not investigated the possibility of using supervised learning, i.e. extracting the subgoal parameters from a given trajectory. However, we think that this should not pose a big problem.

7.4.5 Training Data

The dynamic model of the system can be learned with sample trajectories of the states and the applied controls. On the level of the subgoals the training data is given by the used sub-goal parameters and a corresponding performance evaluation of these sub-goal parameters (RL setup).

7.4.6 Generalization

The AICO algorithm drives the system to the next subgoal in an optimal way (energy efficient). Thus, the approach generalizes easily to different initial states of the system, at least if the subgoal is still reachable from the new initial state. Thus, for simple control problems like kinematic reaching tasks, the generalization of the approach to different initial states is inherent to the approach. For complex control problems, where different subgoals have to be used for different initial conditions, the approach should generalize well due to the abstract representation of the movement.

7.4.7 Modulation

The trajectory is modulated by the cost function. For example smaller control costs will automatically result in an faster movement execution. In addition the speed of the trajectory can be easily modulated by using different duration parameters d_i. Changing the subgoal will result in different trajectories.

7.4.8 Sensory Feedback Integration

The feedback-controller can (in theory) deal with arbitrary Gaussian noise in the system dynamics. Including additional sensory feedback at the level of the feedback controller is difficult because we would need an exact model of the sensors and of the environment. However, sensory feedback can be integrated at the level of the subgoal parameters, but it is not completely clear how to do that.

7.4.9 State Variables

The state variables are given by the state of the robot (depending on the problem joint-positions or also joint-velocities) and on the current execution time of the template.

7.4.10 Robustness and Adaptation to Perturbations

The feedback-controller can (in theory) deal with arbitrary Gaussian noise in the system dynamics. The feedback controller always tries to track the mode of the posterior trajectory distribution in an optimal way (according to the defined costs). Unexpected perturbations, which require a re-planning of the sub-goal parameters, need to be detected by a separate module.

7.4.11 Stability

The controller is stable in a local neighborhood of the posterior belief of the trajectory.

7.5 Non-Functional Analysis

7.5.1 Representation and Interface

The motion is represented as a sequence of subgoals. The number of subgoals is usually fixed before learning.

7.5.2 Timing

Time is a separate variable of the system, determining how much time is left to reach a subgoal. Replacing the time variable with a phase variable which can be coupled with external events like for the DMPs [46] should be feasible, but has not been investigated.

7.5.3 Robustness and Reliability

For perturbations which do not require re-planning of the subgoals (typically the case) the system is very robust due to the feedback controllers. Re-planning of the subgoals in real time will be difficult due to computational reasons.

7.5.4 Dependencies

The system depends on the quality of the knowledge of the system dynamics. The simulation time step used for planning is also quite crucial, for highly non-linear systems like the 2-link balancing task, this time-step has to be very small, say 1ms to 5ms.

7.5.5 Runtime

Planning a single trajectory given the subgoal parameters is sufficiently fast and can be done in less than a second even for high dimensional systems.

Learning the subgoal parameters itself is currently implemented by a genetic algorithm (CMA-ES) which is rather slow (can take several hours). However, we used this algorithm for the RL setup which is considered to be much more complicated as the imitation learning setup.

7.5.6 Usefulness for Recognition

If it is possible to extract the subgoals from given trajectories (should be feasible), the subgoal parameters are due to the compact representation well suited for movement recognition.

7.6 Summary

Representing a motion primitive as a sequence of subgoals is a very compact representation of a movement, resulting in a low number of parameters to describe a motion. The hypothesis that the compact subgoal representation facilitates learning with motion primitives seems to be plausible, however, this still needs to be investigated. The subgoal representation also allows the application of local planning methods for complex movement skills which would otherwise not be achievable with these methods.

Learning takes place at two levels of hierarchy. At the motor control level we have to learn the system dynamics. At the level of the movement primitives, good subgoals have to be found in order to achieve a given task. Thus the problem of learning the whole motion with motion primitives is decomposed into two sub problems, which are supposed to be easier to accomplish.

Due to the non-linear feedback controller the approach is also robust to various kinds of perturbations. This property should avoid the need for re-planning the subgoals in the case of unexpected perturbations.

Chapter 8

Adaptive Frequency Oscillators

Sébastien Gay[1], Auke Jan Ijspeert[1], Juan Pablo Carbajal[2], Hidenobu Sumioka[2], Qian Zhao[2], Naveen Suresh Kuppuswamy[2]
EPFL, BIOROB[1], UZH[2]

8.1 Short Introduction

Adaptive Frequency Oscillators (AFOs) extend standard Central Pattern Generators (CPGs) to allow the system to learn the frequency of a periodic input (*forcing*) signal and adapt its own intrinsic frequency to it. The frequency tuning of Adaptive Frequency Oscillators goes beyond mere synchronization or entrainment as in generic CPGs. The system is called *plastic* in that it adapts its own parameters to learn the frequency of the input signal. This frequency remains encoded in the system even if the forcing signal disappears.

Adaptive Frequency Oscillators achieve the following:

1. All characteristics of standard CPGs: no explicit time dependence, stability, low computational cost etc. (refer to the section on CPGs for more precisions).
2. They learn the frequency of any periodic (harmonic or non-harmonic) input signal: sine wave, square pulse, sawtooth etc. When the forcing signal encapsulates several frequency components the AFO learns the frequency component that is the nearest to its initial frequency.
3. They can track a frequency that is changing with time.
4. They can tune themselves to the resonant frequency of the robot by providing the information of proper sensors as forcing signal.
5. They can split the different frequency components of a complex signal by using a pool of AFOs with different initial conditions. The different oscillators can then be recombined to get a pattern generator with the same complex waveform as the input signal.

Some parts of the text and the figures of this documents have been taken from [47].

8.2 Model Description

The idea of Adaptive Frequency Oscillators is to add a learning rule on the frequency to an existing oscillator so that it learns the frequency of a periodic input signal. In

this document, we will discuss mostly the Adaptive Hopf Oscillator, but the learning rule presented here can be applied to other oscillators as well.

The main idea behind Adaptive Frequency Oscillators is the following: Starting from a standard Hopf oscillator written in Cartesian coordinates and with its output x perturbed by a periodic input signal F(t):

$$\begin{aligned} \dot{x} &= \gamma(\mu - (x^2 + y^2))x - \omega y + \epsilon F(t) \\ \dot{y} &= \gamma(\mu - (x^2 + y^2))y + \omega x \end{aligned} \tag{8.1}$$

where μ defines the squared radius of the limit cycle of the oscillator, $\gamma > 0$ influences the speed of convergence to this limit cycle, and ϵ defines the strength of the coupling with the input signal.

Let us write this system into polar coordinates to investigate the influence of the forcing signal on the phase of the oscillator. We pose $x = r\cos(\phi)$ and $y = r\sin(\phi)$:

$$\begin{aligned} \dot{r} &= \gamma(\mu - r^2)r + \cos(\phi)\epsilon F(t) \\ \dot{\phi} &= \omega - \frac{1}{r}\sin(\phi)\epsilon F(t)) \end{aligned} \tag{8.2}$$

Since we want to find a learning rule for ω which drives it towards the frequency of the perturbation, one should impose the same effect on the frequency (i.e. accelerate or decelerate) as the forcing signal imposes on the phase (i.e. the tangent component of the forcing term). As shown above, and considering that the system has converged to its limit cycle ($r = const$), the learning rule is chosen to be:

$$\dot{\omega} = -\frac{1}{\tau}\sin\phi\epsilon F(t)) \tag{8.3}$$

where τ is a factor influencing the convergence rate of ω to the desired frequency.

Switching back to Cartesian coordinates the full system is written:

$$\begin{aligned} \dot{x} &= \gamma(\mu - (x^2 + y^2))x - \omega y + \epsilon F(t) \\ \dot{y} &= \gamma(\mu - (x^2 + y^2))y + \omega x \\ \dot{\omega} &= -\frac{\epsilon}{\tau}\frac{y}{\sqrt{x^2 + y^2}}F(t) \end{aligned} \tag{8.4}$$

This learning rule on ω can be similarly applied to different oscillators like amplitude controlled phase oscillator, Van Der Pol or Fitzhugh-Nagumo oscillator. (See [47])

Adaptive Frequency Oscillators have mainly two different applications:

- They can tune themselves to the resonance frequency of a robot with passive dynamics, making the locomotion very efficient.
- A pool of AFOs can be used to perform frequency analysis on an input signal. The different AFOs of the pool can then be combined to construct limit cycles with arbitrary shapes.

Tuning to the resonance frequency of a robot is simply achieved by providing the proper sensor signal (ex: from load sensors on the feet or potentiometers from the joints) to the AFO. The system automatically tunes itself to the resonant frequency of the system and tracks changes of resonance frequency (if one adds or remove weight

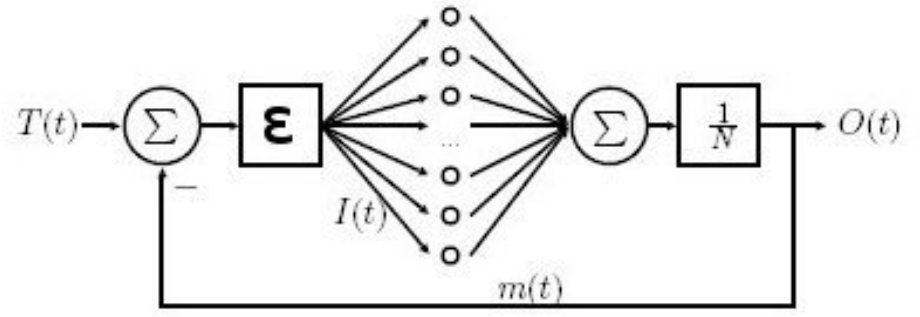


Fig. 8.1: Structure of the pool of adaptive frequency oscillators that is able to reproduce a given teaching signal T(t). The mean field produced by the oscillators is fed back negatively to the oscillators

for instance). See [48] for an example with a compliant quadruped robot.

Performing frequency analysis on complex input signals with multiple frequency components requires the use of a pool of AFOs with a negative feedback loop, as described on Figure 8.1.

Each AFO will tune itself to the frequency that is the nearest to its initial frequency. Each time an oscillator converges to a frequency present in the teaching signal, due to the negative feedback, this frequency disappears from the input signal. The other oscillators can then only converge to the other frequencies present. The pool of AFOs after convergence is able to approximate the frequency spectrum of the teaching signal.

The previous pool of oscillators can be extended by adding a weight to each oscillator in the mean field sum, and a coupling between oscillators, in order to ensure stability of the output pattern. The result is an individual oscillator will be able to fully match the energy content of a frequency in the spectrum of the teaching signal. Moreover, the coupling ensures that the system exhibits a stable limit cycle. Here, amplitudes and phase differences become system state variables, in addition to frequencies. The governing differential equations of the system are then, for oscillator i:

$$\begin{aligned}
\dot{x_i} &= \gamma(\mu - ({x_i}^2 + {y_i}^2))x_i - \omega_i y_i + \epsilon F(t) + \xi \sin(\frac{\omega_i}{\omega_0}\phi_0 - \phi_i - \psi_i) \\
\dot{y_i} &= \gamma(\mu - ({x_i}^2 + {y_i}^2))y_i + \omega_i x_i \\
\dot{\omega_i} &= -\frac{\epsilon}{\tau}\frac{y_i}{\sqrt{{x_i}^2 + {y_i}^2}}F(t) \\
\dot{\psi_i} &= \sin(\frac{\omega_i}{\omega_0}\phi_0 - \phi_i - \psi_i) \\
\dot{\alpha_i} &= \eta x_i F(t)
\end{aligned} \tag{8.5}$$

with

$$\begin{aligned}
\phi_i &= \mathrm{sgn}(x_i)\cos^{-1}(-\frac{y_i}{\sqrt{{x_i}^2 + {y_i}^2}}) \\
F(t) &= P_{teach}(t) - Q_{learned}(t) \\
Q_{learned}(t) &= \sum_{i=0}^{N} \alpha_i x_i
\end{aligned}$$

where ξ , K and η are positive constants. The output of the system, $Q_{learned}$, is the weighted sum of the output of each oscillator. $F(t)$ represents the negative feedback, which on average is the remainder of the teaching signal $P_{teach}(t)$ that the network still has to learn. α_i represents the amplitude associated with the frequency ω_i of oscillator i. The evolution equation maximizes the correlation between x_i and $F(t)$, which means that α_i will increase only if ω_i has converged to a frequency component of $F(t)$ (the correlation will be positive on average) and will stop increasing when the frequency component ω_i disappears from F(t) because of the negative feedback loop. ψ_i is the phase difference between oscillator i and 0. The value converges to the phase difference between the instantaneous phase of oscillator 0, ϕ_0, scaled for frequency ω_i, and the instantaneous phase of oscillator i, ϕ_i. Each adaptive oscillator is coupled with oscillator 0, with strength ξ, to maintain the correct phase relationships between oscillators. See [47] for additional information.

8.3 Simulation

8.3.1 General Results

We first present results of numerical simulation where we impose various kinds of teaching signals to the AFO. Figure 8.2 shows convergence of the AFO to the frequency of the teaching signal for different signal shapes. The AFO tunes itself to the correct frequency even for non harmonic signals (b and c). It can also track a changing frequency (d) tunes itself to one of the frequency component of a multi-frequency input signal. AFOs can even approximate the frequency of a chaotic signal like the output of the Rössler oscillator (f).

8.3.2 AFOs and Energy Transfers with Mechanical Systems

As shown in [49] the convergence of ω in the system (Eq.(8.4)) depends on the coupling parameter ϵ: the higher this parameter the faster the convergence, but the quality of the convergence becomes low. Figure 8.3 shows the minimum value of ${}^{(\omega_2-\omega_1)}/_{\omega_1}$ for which the AFO is able to separate the frequencies in the spectrum of the input signal $F(t) = \frac{1}{2}\left(sin(\omega_1 t) + sin(\omega_2 t)\right)$. As can be seen, the performance of the separation becomes low with increasing coupling parameter, but the time required to converge to the frequency is shorter, defining a trade-off between speed of convergence and separation performance.

AFOs can be coupled with mechanical systems and can tune themselves to the natural frequency of these systems. However, the energy transfer between the mechanical system and the AFO that is possible to achieve is largely dependent of the properties of the system. Figure 8.4 shows the performance of the AFO when coupled, as described in Section 8.2, to a linear damped mass-spring system defined as:

$$\begin{aligned} \dot{z} &= v \\ \dot{v} &= -(2\pi f_0)^2 z - dv + \cos(\omega t) \quad (f_0 = \frac{1}{2\pi}\sqrt{\frac{k}{m}},\ d = \frac{\gamma}{m}) \end{aligned} \tag{8.6}$$

where $f_0 = 4.8Hz$ and $d = 0.51/s$ (we observed a similar tendency for softer damped mass-spring system ($f_0 = 3.0Hz$, $d = 0.0011/s$)). The quality index $Q(Q \in [0, 1])$ quantifies the performance of the AFO in terms of energy transfer. The performance in terms of energy transfer (Q) was largely poor with the most number of

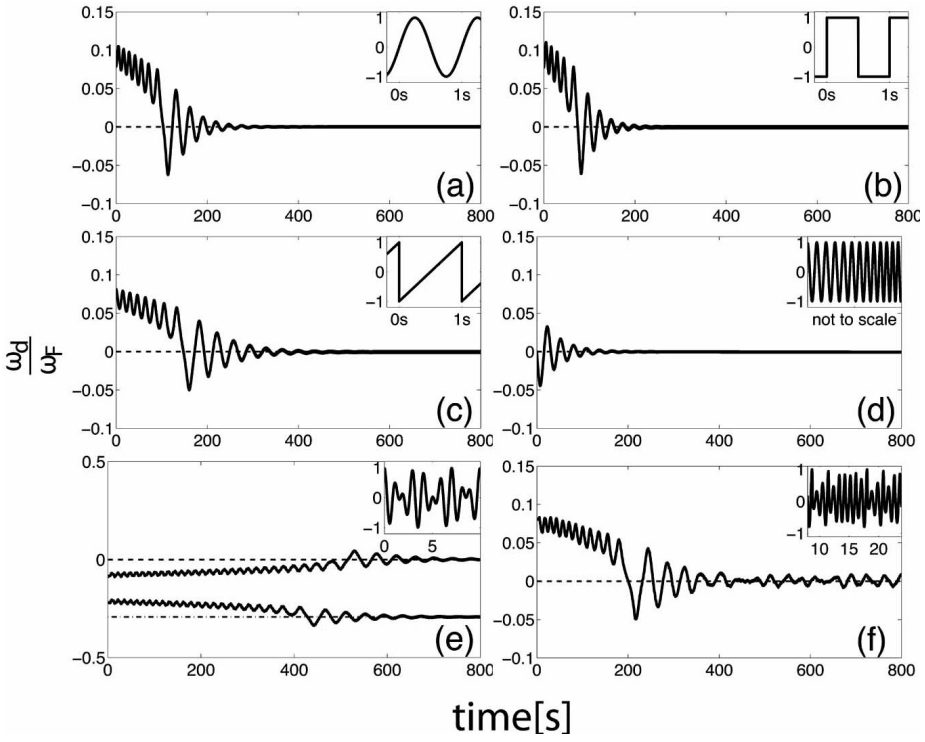


Fig. 8.2: (a) Typical convergence of an adaptive frequency Hopf oscillator driven by a harmonic signal ($F(t) = sin(2\pi t)$). The frequencies converge towards the frequency of the input (indicated in dashed line). After convergence the frequency oscillates with a small amplitude around the frequency of the input. In all figures, we plot in the main graph the time evolution of the difference between ω and the input frequency ω_F, normalized by the input frequency. The top right panel shows the driving signals (note the different scales). (b) Square pulse $F(t) = rect(\omega_F t)$ (c) Sawtooth, $F(t) = st(\omega_F t)$, (d) Chirp F(t) = $\cos(\omega_c t)$ where $\omega_c = \omega_F(1+\frac{1}{2}(\frac{t}{1000})^2)$. (Note that the graph of the input signal is illustrative only since changes in frequency takes much longer than illustrated). (e) Signal with two non-commensurate frequencies $F(t) = \frac{1}{2}[\cos(\omega_F t)+\cos(\frac{\sqrt{2}}{2}]$, i.e. a representative example of how the system can evolve to different frequency components of the driving signal depending on the initial condition $\omega_d(0) = \omega(0) - \omega_F(0)$ (f) F(t) is the non-periodic output of the Rössler system. The Rössler signal has a $\frac{1}{f}$ broad-band spectrum, yet it has a clear maximum in the frequency spectrum. In order to assess the convergence we use $\omega_F = 2\pi f_{max}$, where f_{max} is found numerically by FFT. The oscillator converges to this frequency.

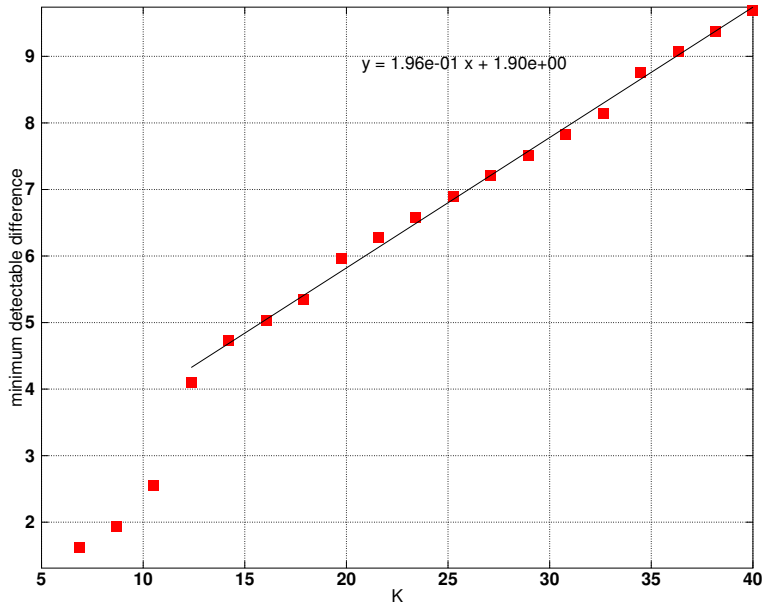


Fig. 8.3: The minimum values of $^{(\omega_2-\omega_1)}/_{\omega_1}$ (all ω_s are expressed in rad/s) that allow the AFO to separate ω_1 or ω_2 (here, $\omega_1 = 20\pi$, $\omega_2 \in [1.01\omega_1, 1.1\omega_1]$) from the input signal $F(t) = \frac{1}{2}\left(sin(\omega_1 t) + sin(\omega_2 t)\right)$ for different values of coupling parameter. The initial phase is set to zero and the initial radius of the oscillator is 1. We take the initial frequency of the oscillator uniformly at the range, $\omega \in [0.9\omega_1,\ 1.1\omega_2]$. As can be seen, the AFO with the lower parameter can separate one of frequency components of the inputs even when $^{(\omega_2-\omega_1)}/_{\omega_1}$ value is small. However, it takes longer time to converge as shown in [49].

initial conditions on ω having $Q \sim 0.2$ and only initial conditions with $\epsilon = 20$ achieve stabilisation with some of the ω converging to the natural frequency of the mechanical system.

However, when coupled to a nonlinear spring, the performance of the system is quite different. The previous spring was extended with the third power of the deformation:

$$\begin{aligned} \dot{z} &= v \\ \dot{v} &= -az^3 - (2\pi f_0)^2 z - dv + \cos(\omega t) \end{aligned} \tag{8.7}$$

This time, the performance is surprisingly high ($Q \simeq 0.5$ averaged over all initial conditions). Figure 8.5 shows the evolution of the amplitude and frequency of the AFO. We notice that for the linear case the frequency converges to values very close to f_0, nevertheless no amplification of the oscillations is observed. The evolution of the frequency in the non-linear case shows a strong variation which we presume is the source of the resonant pump of energy into the system. In contrast with the linear system, the non-linear one shows an increase of the amplitude.

8.3.3 AFOs and Compliant Robotics Systems

AFOs can be coupled to real compliant robots and tune themselves to the resonant frequency of the whole robot. Unlike above, the AFOs cannot be coupled to the actual oscillations of the robot (which is unknown a priori), but has to be coupled to some sensory information. Figure 8.6 shows a small compliant quadruped robot, the puppy, controlled by a single AFO coupled to the vertical axis of the inertia sensor of the robot. The AFO is able to tune itself to the resonance frequency of the robot and track changes of resonance frequency (by adding or removing weight). It is also capable of adapting to the resonance frequency of a specific posture (here by changing the leg angle). Convergence has however been proven dependent to the type of sensor used and particularly its phase shift with respect to the general oscillations of the robot. For a theoretical discussion and experiments about convergence with different types of sensors, please refer to [48].

8.3.4 Pools of AFOs for frequency analysis and construction of complex limit cycles

Finally pools of AFOs can be used to find the frequency components of a multi-frequency input signal. Figure 8.7 shows an example of a frequency analysis and construction of a limit cycle using a pool of oscillators. The evolution of the frequency ω, the amplitude α and the phase ϕ of each oscillator is represented. Each oscillator eventually converges to one frequency of the teaching signal and the pool approximates the input signal perfectly. See [47] and [50] for more details.

8.4 Functional Analysis

8.4.1 Dynamics and Nonlinearity

The dynamics of the oscillator on which the AFO is built is kept by adding the learning rule on the frequency parameter. The oscillator shape and dynamics do not change.

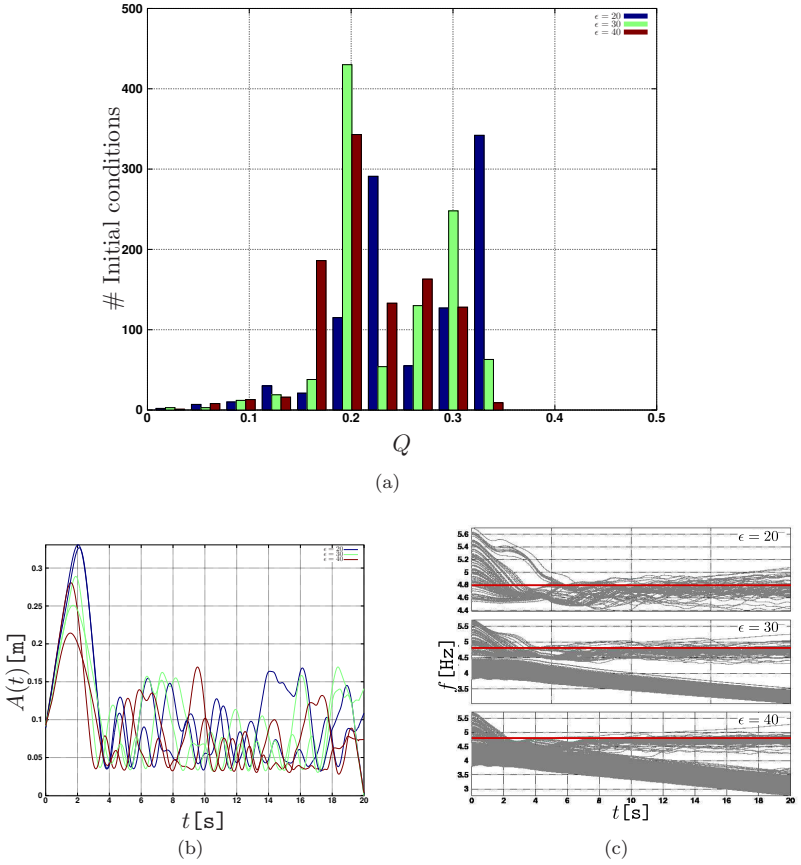
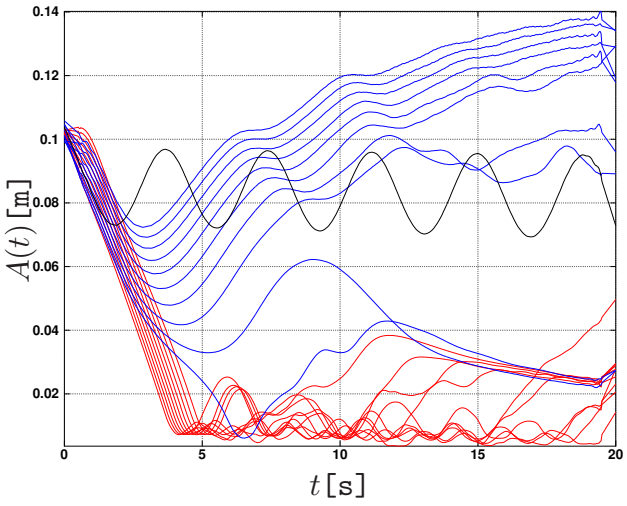
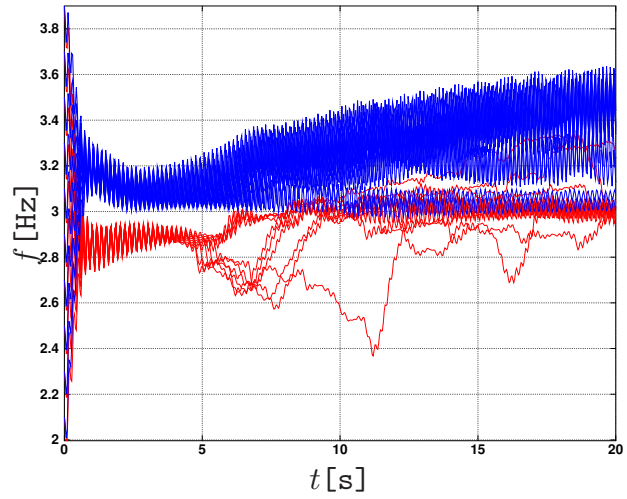


Fig. 8.4: The behavior of the damped system coupled with the AFO for three coupling parameter values ($\epsilon = 20, 30, 40$). The initial conditions of the AFO are radius $r = 0.01$, phase $\phi = 0$, and $f \in [0.8f_0, 1.2f_0]$. For each parameter setting, the simulation is run for 20s at 1000 initial conditions. a) Histogram of index Q. b) Amplitude of the oscillations of the spring for the initial conditions where we get the highest Q value and the lowest Q one for different coupling parameters ϵ. In both conditions, the amplitude often becomes lower than the initial one. c) Time series of the frequency of the AFO. Each gray line represents time series of a frequency for an initial condition. The natural frequency of the system is shown with red color. The lowest coupling parameter ($K = 20$) slightly improves the adaptation of the AFO to the natural frequency.



(a)



(b)

Fig. 8.5: The behavior of an AFO connected to a progressive spring and that of an AFO connected to a linear spring ($\epsilon = 500$ for both of them). The time evolution of the instantaneous amplitude a) and of the frequency of the AFO b) are shown. The blue lines correspond to the nonlinear spring, the red lines to the linear spring and the black line shows the evolution of the amplitude of the non-linear spring when forced with a cosine function with frequency equal to the linear resonant frequency $f_0 = 3$Hz. The non-linear case in blue lines presents an average performance of $Q \simeq 0.5$ due to the initial drop in the amplitude. It is clear that the resulting amplitude is higher in the case of an AFO coupled to a nonlinear spring than when the same nonlinear spring is forced by a simple cosine function (in black). Note that in b) the frequency of the AFO, for the linear case, is close to f_0, unlike in the nonlinear case, nevertheless resonance in the amplitude is not observed.

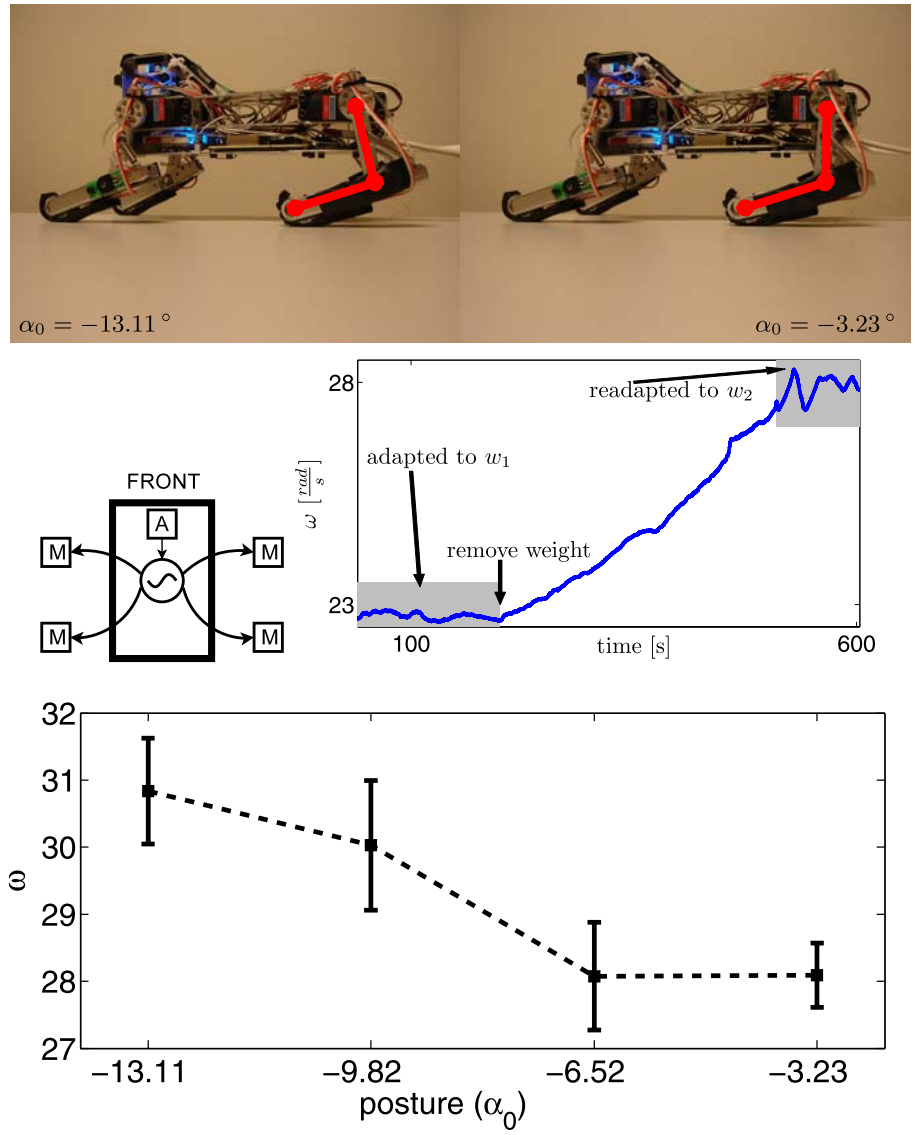
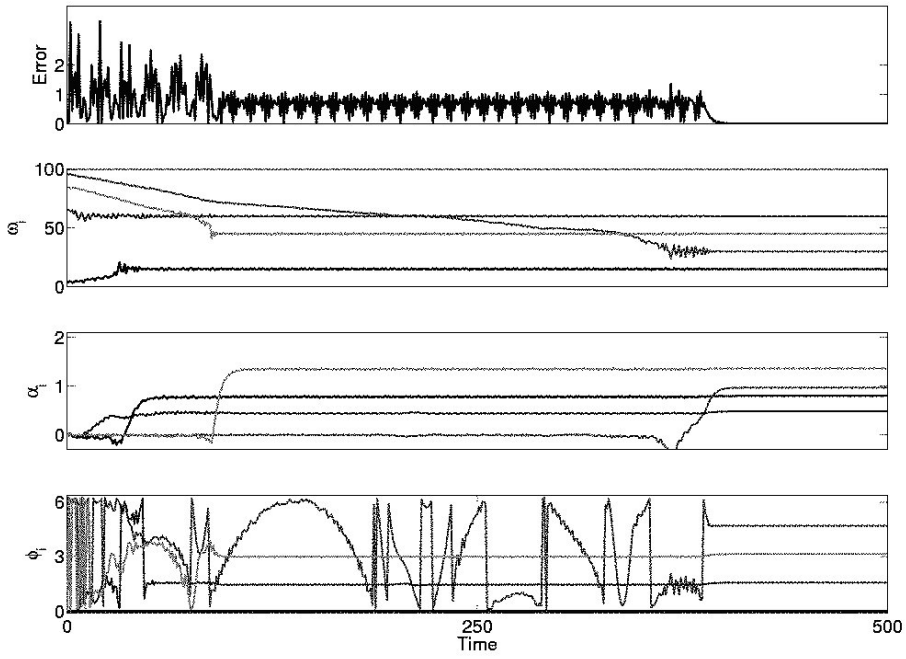
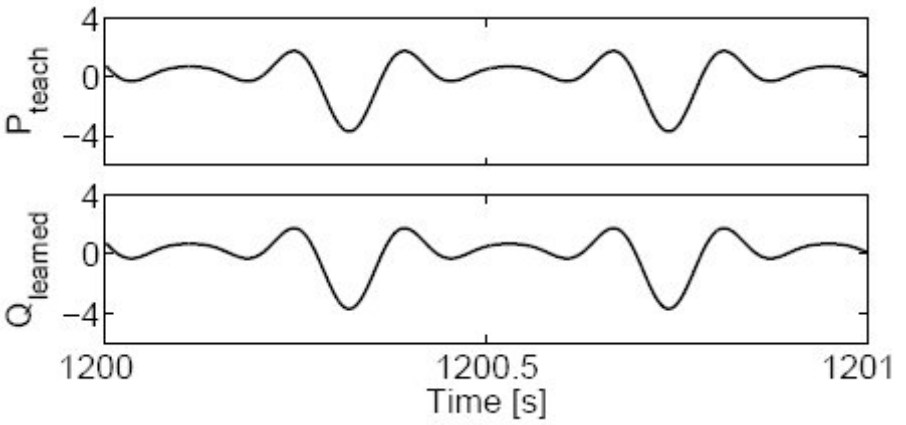


Fig. 8.6: Online adaptation of AFOs controlling the hip joint of a simple compliant quadruped robot. Top: the quadruped robot used for the experiments with two different leg angles. The compliance is in the knees and only the hips are actuated. Middle left: illustration of the sensor-controller structure. The z-value of the acceleration sensor (A) is used as an input to the oscillator. The signal of the oscillator is used to set the motor position according to. Middle right: intrinsic frequency ω of the oscillator shown as it evolves during the experiment. The body weight is changed from m1 = 0.905 kg to m2 = 0.695 kg. The controller immediately starts to adapt to the changed body property Bottom: the average frequency (squares) found by the adaptive frequency oscillator vs. posture, i.e. the angle of the leg. The bars show the standard deviation.



(a) Evolution of the state variables of the system



(b) Result of learning

Fig. 8.7: Construction of a limit cycle by learning an input signal ($P_{teach} = 0.8\sin(15t)+\cos(30t)-1.4\sin(45t)-0.5\cos(60t)$). Fig. (5a) shows the evolution of the state variables of the system during learning. The upper graph is a plot of the error ($||P_{teach} - Q_{learned}||$). The 3 other graphs show the evolution of the frequencies, ω_i, the amplitudes, α_i and the phases, ϕ_i. We clearly see that the system can learn the teaching signal perfectly – the frequencies, amplitudes and phase differences converge to the correct values and the error becomes zero. Fig. (5b) shows the result of learning (teaching signal in upper graph, output of the system in lower graph), we note the perfect reconstruction of the signal.

8.4.2 Attractor

The system is designed to learn the frequency of periodic signals. Thus it can be applied to oscillators with limit cycle behaviors. The frequency of the input signal is a global attractor of the system (infinite basin of attraction), although the frequency of the oscillator never quite stays at the teaching frequency but keeps oscillating around it with very small amplitude.

8.4.3 Coupling

For online learning of the resonance frequency of a robot, the AFOs can be coupled in the same way as standard CPGs (see Section on CPGs). Usually, one AFO per DoF is used. For frequency analysis and construction of limit cycles, several AFOs are coupled together as described in Section 8.2. This time, one AFO per frequency component is used and this pool of AFO forms a higher dimensional limit-cycle, a meta-oscillator. One can then use these new modules for locomotion the same way as standard CPGs, by using one per DoF and coupling them together to have specific phase relationships.

8.4.4 Learning

Type AFOs are a type of supervised learning since a teaching signal is provided.

Algorithm All the learning is embedded in the dynamics of the system. No external learning algorithm is used, although AFOs can be considered as a type of Hebbian learning.

Mode AFOs are well suited for online learning, especially to tune themselves to the resonant frequency of the robot, since the learning is based on the information provided online by the various sensors. They can also track a varying frequency online. Pools of AFOs can be used offline to construct a limit cycle based on some training signal and then applied to the robot and modulated online.

8.4.5 Training Data

The training data can be any periodic or pseudo-periodic signal, depending on the application. For resonance frequency tuning AFOs use sensor outputs. Note: the phase relationship between the sensor and the oscillator output must respect some bands to avoid divergence (see [48]). This might require shifting the phase of the sensor. For frequency analysis, a desired teaching signal has to be provided. It can be for instance the desired trajectory of one joint.

8.4.6 Generalization

AFOs are capable to generalize to any periodic signal with any frequency. The basin of attraction of the frequency is infinite. The learning is made online so AFOs can adapt online to changes in the frequency.

A pools of AFOs coupled together is capable of generating a limit cycle modeling any teaching signal, as long as the number of AFOs chosen to compose the pool is at least equal to the number of frequency components of the teaching signal. If fewer oscillators are used, or if the spectrum is continuous, the output will only approximate the teaching signal.

8.4.7 Modulation

In addition to the base oscillator parameters, the coupling terms ϵ and convergence speed τ can be modulated, both influencing the convergence rate of the frequency to the input frequency as well as the amplitude of the remaining oscillations of the frequency after convergence.

8.4.8 Sensory Feedback Integration

Adaptive Frequency Oscillators are intrinsically suited for sensory feedback integration. Indeed, one of their main application is to be able to learn the resonance frequency of a mechanical system. For this, information from well chosen sensors (see note in Section 8.4.5) is provided to the AFO as input signal, and the system automatically tunes itself to the resonance frequency of the system. Other kinds of sensory feedback information can be imagined, similarly to CPGs.

8.4.9 State Variables

Depending on the oscillator that is used as basis for the AFO, different state variables may be used. For the Hopf oscillator in Cartesian coordinates, two coupled state variables x and y are used, x being generally chosen as the output of the system. In addition to the intrinsic state variables of the base oscillator, the frequency, ω, is made a state variable of the system to allow learning of the input frequency and not only entrainment.

8.4.10 Robustness and Adaptation to Perturbations

The characteristics the oscillator on which the AFO is built are kept, thus the robustness of the limit cycle is the same. For more precisions on characteristics of some well known oscillators, please see the section on CPGs. Perturbations on the frequency are immediately damped out by the system and the frequency converges back to that of the input signal. The basis of attraction of the frequency being infinite, this behavior is independent of the amplitude of the perturbation, although the time to recover may change.

8.4.11 Stability

Again, stability of the limit cycle of the oscillator depends of the oscillator used (see section on CPGs). Convergence of the frequency to that of the teaching signal has been proven for small coupling term $\epsilon << 1$ in [49] and for big $\epsilon >> 1$ in [50].

8.5 Non-Functional Analysis

8.5.1 Representation and Interface

AFOs can take as input any periodic signal. They output trajectories the same way standard oscillators do, but with the frequency of these oscillations tuned to the frequency of the input signal. The input signal can even be a multi-frequency signal in which case the AFO will tune itself to the frequency component which is the nearest to its initial frequency. In the case of programmable central pattern generators, using pools of AFO, the number of AFOs has to be predefined. The output is a limit cycle which shape approximates that of the input signal. It the number of AFOs is at least equal to the number of frequency components of the input signal, the shape of the limit cycle will perfectly match that of the teaching signal. The approach is suitable

to be used as an independent module since inputs, outputs and modulation commands are well separated.

8.5.2 Timing

The limit cycle part of AFOs work on the same timing as the oscillator it is based on. These oscillators usually have an explicit or implicit phase definition, which acts as an internal clock. Although no explicit time dependence is used, the dynamical systems are integrated with some predefined time-step. See section on CPGs for more information.

The convergence of the frequency of the AFO to the frequency of the input signal works on a slower timescale than the convergence of the trajectory to the limit cycle. This timescale can however be modulated to some extend by changing the coupling terms ϵ and τ.

8.5.3 Robustness and Reliability

AFOs can give Anytime guarantee: the input signal can be changed at anytime, leading to the AFO tuning itself to the frequency of the new signal. In addition to the parameters of the oscillator defining the limit cycle, AFOs have two tunable parameters ϵ and τ. Since the coupling term ϵ is applied to x as well as ω, setting a too big ϵ can increase the amplitude of the oscillations. Setting a too small τ, however, can lead to instabilities in the frequency.

8.5.4 Dependencies

AFOs are intrinsically coupled to an external signal. This signal can come from sensors in particular to learn the resonance frequency of a mechanical system (See however the note in section 8.4.5 about the choice of sensor). This signal can however be any kind of signal, when using pools of AFOs for frequency analysis for instance. It can be predefined joint trajectories when using pools of AFOs as programmable central pattern generators.

8.5.5 Runtime

AFOs, like oscillators, do not have well defined states since everything (trajectory generation, frequency learning) is embedded in the dynamics of the system. As a matter of fact, even when the frequency seem to have converged to the input frequency, it actually keeps on oscillating around it with very small amplitude.

The complexity of a single AFO is of the order of the integrator, and no additional computation loop is needed. The complexity is comparable to that of a standard oscillator.

The number of integration steps to generate an arbitrary shaped limit cycle using pools of AFOs is dependent of the following factors:

- The number of AFOs used (which depends of the number of frequency components in the input signal to approximate).
- The initial frequency chosen for each oscillator. The higher the difference between the initial frequency of each AFOs and the frequency component to converge to, the longer the convergence time.

8.6 Summary

In this chapter we presented the Adaptive Frequency Oscillators, which are an extension to standard oscillators enabling them to adapt their frequency to that of an input signal. AFOs add a simple learning rule to the frequency of the oscillator, which becomes a state variable. All the learning is embedded in the dynamics of the oscillator. Two main applications exist for AFOs. First, when coupling an AFO with some well chosen sensor, the system is able to tune itself to the resonance frequencies of the mechanical system. Convergence and performance in terms of energy transfer is however dependent on the coupling parameter used as well as the type of mechanical system. Coupling AFOs with a compliant legged robot allows for efficient locomotion, adaptation to changes of body properties and postures. Second, using a pool of AFOs, it is possible to find the different components of a multi-frequency signal, which is useful for frequency analysis for instance. The different AFOs can then be recombined to generate an arbitrary shaped limit cycle. This is very useful to program a limit cycle with a predefined complex shape, which can be modulated afterwards.

Chapter 9

Neural Central Pattern Generator

Francis Wyffels, Benjamin Schrauwen
UGent

9.1 Short Introduction

In this chapter we introduce a new methodology for CPG design based on Reservoir Computing. We call the constructed modules Neural Central Pattern Generators (NCPGs) because they use a recurrent neural network, the reservoir, to autonomously generate rhythmic patterns. For training, a recently published paradigm called FORCE learning [51], is used to adjust the output weights of our system. Apart from generating high-dimensional complex rhythmic patterns in a stable and robust way, the system's output can be modulated using low dimensional control inputs or adjusting some parameters of the NCPG. Sensor information can be incorporated in our system by feeding it to additional inputs.

9.2 Model Description

The core technique used for the Neural Central Pattern Generator (NCPG) model is Reservoir Computing [52–54]. Basically, we use a large recurrent neural network, the reservoir, of which only the output weights are trained using standard linear regression techniques. The reservoir is composed of randomly connected sigmoid neurons. During both training and testing, the neuron states are updated using the following equation:

$$\mathbf{x}[k+1] = (1-\lambda)\mathbf{x}[k] + \lambda \tanh\Big(\mathbf{W}^{\text{res}}_{\text{res}}\mathbf{x}[k] + \mathbf{W}^{\text{res}}_{\text{inp}}\mathbf{u}[k] + \mathbf{W}^{\text{res}}_{\text{out}}\mathbf{y}[k] + \mathbf{W}^{\text{res}}_{\text{bias}}\Big) \quad (9.1)$$

The states $\mathbf{x}[k+1]$ at time step $k+1$ depend on the states $\mathbf{x}[k]$ at time step k, an additional input $\mathbf{u}[k]$ (can be a control input but also sensor feedback), the output of the system $\mathbf{y}[k]$ and a bias. By changing the leak-rate λ, the system's dynamics can be tuned effectively. In our setup we use sigmoidal neurons but other non-linearities might also be used. Initially, all weights $\mathbf{W}^{\text{res}}_{\star}$ are created randomly (usually drawn from uniform or normal distributions) and are fixed during both training and testing. The weight matrix $\mathbf{W}^{\text{res}}_{\text{res}}$ is drawn from a normal distribution with zero mean and unit variance. A large part of the weights is set to zero according the used connection fraction c_{res}. After construction, the weight matrix $\mathbf{W}^{\text{res}}_{\text{res}}$ is rescaled such that the

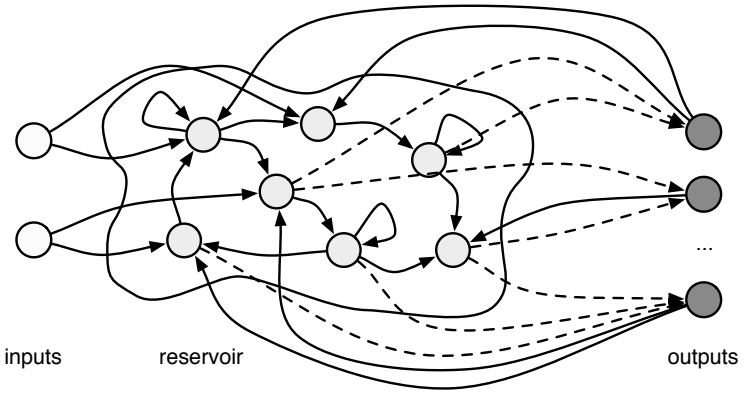


Fig. 9.1: Schematic overview of the NCPG model. The system has one or more outputs which are fed back into the reservoir. Additional control inputs can be used for modulation of the output signals. Only connections directed to the output, denoted by dashed lines, are trained. The other connections are initially randomly created and remain unchanged during training/testing.

largest eigenvalue (i.e., the spectral radius) is equal to 1.5. This causes the reservoir to be chaotic. This differs from the usual approach where the spectral radius is chosen such that the system operates at the edge of chaos. We will use FORCE learning to train the reservoir system, which require the reservoir to spontaneously generate some activity. Apart from this difference, it is important to note that the output of the system is fed back during both training and testing. This is different from the echo state networks approach, in which the desired output is fed back (teacher forced) during training.

The weights $\mathbf{W}_{\text{res}}^{\text{out}}$ are trained online using FORCE learning. This means that Recursive Least Squares is used. For this, during training, every time step the reservoir-to-output weights $\mathbf{W}_{\text{res}}^{\text{out}}$ and the output is adjusted according following equations [51]:

$$\mathbf{y}[k+1] = \mathbf{W}_{\text{out}}^{\text{res}}\mathbf{x}[k+1] \tag{9.2}$$

$$\mathbf{e}[k+1] = \mathbf{y}[k+1] - \mathbf{y}_{\mathbf{desired}}[k+1] \tag{9.3}$$

$$\mathbf{P} = \mathbf{P} - \frac{\mathbf{P}\mathbf{x}[k+1]\mathbf{x}^{\mathbf{T}}[k+1]\mathbf{P}}{1 + \mathbf{x}^{\mathbf{T}}[k+1]\mathbf{P}\mathbf{x}[k+1]} \tag{9.4}$$

$$\mathbf{W}_{\text{out}}^{\text{res}} = \mathbf{W}_{\text{out}}^{\text{res}} - \mathbf{e}[k+1]\mathbf{P}\mathbf{x}[k+1] \tag{9.5}$$

Here $\mathbf{y}_{\mathbf{desired}}$ is the desired output of the system, e.g., an example trajectory, and $\mathbf{e}[k+1]$ is the error at time step $k+1$. $\mathbf{P}$ is an $N \times N$ matrix with N the number of neurons in the reservoir. After training, the weights $\mathbf{W}_{\text{out}}^{\text{res}}$ are kept fixed to test our system. Table 9.1 summarizes all the parameters of our system and their commonly used values[1].

9.3 Simulations

In this section we give some simulation results that illustrate the performance of our system. In Figure 9.2 we show that our system is able to learn rhythmic patterns. The top graph shows the time evolution of the learned MSO pattern ($\sin(0.2k)$ +

[1] Commonly, in the sense that they should be sufficient to do all simulations in this chapter.

Tab. 9.1: Summary of all NCPG parameters. Most parameters are set tuning them roughly by hand. The number of neurons depends mainly on the dimensionality of the output and the number of gaits.

Parameter	Description	Value
N	number of neurons	100 to 2000
λ	leak-rate	0.2
ρ	spectral radius	1.5
c_{res}	reservoir-to-reservoir connection fraction	0.1
c_{in}	in-to-reservoir connection fraction	1
c_{out}	out-to-reservoir connection fraction	1
c_{bias}	bias-to-reservoir connection fraction	1
β	bias weight variance	0.5
ω	output feedback scale	1
α	learning rate, determines initialization of $\mathbf{P_{init}} = \frac{\mathbf{I}}{\alpha}$	0.1

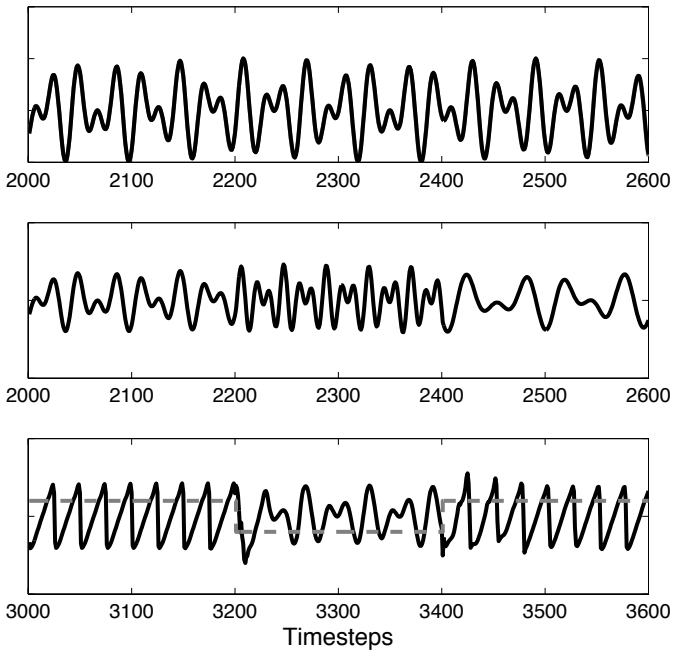


Fig. 9.2: This figure shows the ability to generate a learned rhythmic trajectory. In the top graph one can see, in black, the output of our system after training. The teacher trajectory is $\sin(0.2k)+\sin(0.311k)$ which can be seen in light gray. The second graph shows the ability to modulate the frequency of the output signal by changing the leak-rate parameter of the reservoir system. Every 200 time steps, the leak-rate was set abruptly to an other random value. In the bottom graph we illustrates the ability to switch smoothly from one learned trajectory $\frac{(k \bmod 25)+0.5\sin(0.2k)-12.5}{5}$ to an other $\sin(0.2k)+\sin(0.311k)$. For this, our system was trained using one additional input denoting which trajectory had to be learned.

$\sin(0.311k)$) after training. The desired pattern, shown in light gray, is hard to distinguish as it coincides almost perfectly with the pattern generated by the NCPG. The central graph shows the ability to modulate the frequency by merely tuning the reservoir's leak-rate parameter. Note that the system was not trained explicitly to generate multiple frequencies. The bottom graph illustrates the ability to switch between different gaits. In this case the system was trained with two different patterns, $\frac{(k \bmod 25)+0.5\sin(0.2k)-12.5}{5}$ and $\sin(0.2k)+\sin(0.311k)$, using an additional input which was set to -1 or 1 depending on the desired output pattern. During testing, this input can be switched to control the generated output of the NCPG. Preliminary experiments show that the number of patterns between which a single NCPG can be trained to switch is limited by the complexity of the patterns, the number of neurons and the dimensionality of the output.

In Figure 9.3 we illustrate the ability to learn multi-dimensional patterns. One can see that the system has learned to generate $\sin(0.15k)$, $\sin(0.3k)$ and $\sin(0.2k) + cos(0.1k)$ simultaneously. In total, the desired output dimension was eight. One can observe that our system is able to cope with multiple frequencies. In the same experiment we tested the robustness of the system. Every 200 time steps one or two randomly chosen outputs are clamped to a random outlier value. We see that the system is able to cope with these perturbations and that the coupling between the several outputs remains stable due to the fact that all outputs are derived from the same (coupled) reservoir. This fact is confirmed by the phase portraits in Figure 9.4, illustrating the coupling between several outputs after recovery from the perturbations.

9.4 Functional Analysis

9.4.1 Dynamics and Nonlinearity

The nonlinearity of the system is defined by the sigmoid function and can be adjusted by adding more or less bias, or by scaling up or down the inputs of the system. The dynamics of the system are defined by the randomly generated recurrent neural network, the reservoir, and can be tuned by the spectral radius. Since we are using a spectral radius of 1.5 the reservoir can be considered chaotic. This causes spontaneous activity of the reservoir which is required by FORCE learning. The inputs (including output feedback) are scaled enough large to tame the dynamics of the reservoir such that the system itself is stable.

9.4.2 Attractor

The attractor of the NCPG is a limit cycle attractor, which is defined by the training signal.

9.4.3 Coupling

In the NCPG model, all outputs are derived from the same system: a coupled recurrent neural network. Because of this, all outputs generated by the NCPG are coupled and stay coupled even under severe conditions, as can be observed from Figure 9.3 and Figure 9.4.

9.4.4 Learning

Type Supervised

Algorithm Only the output weights of our system are changed during training using standard linear regression methods. We apply FORCE learning, a method based on recursive least squares, for training our system.

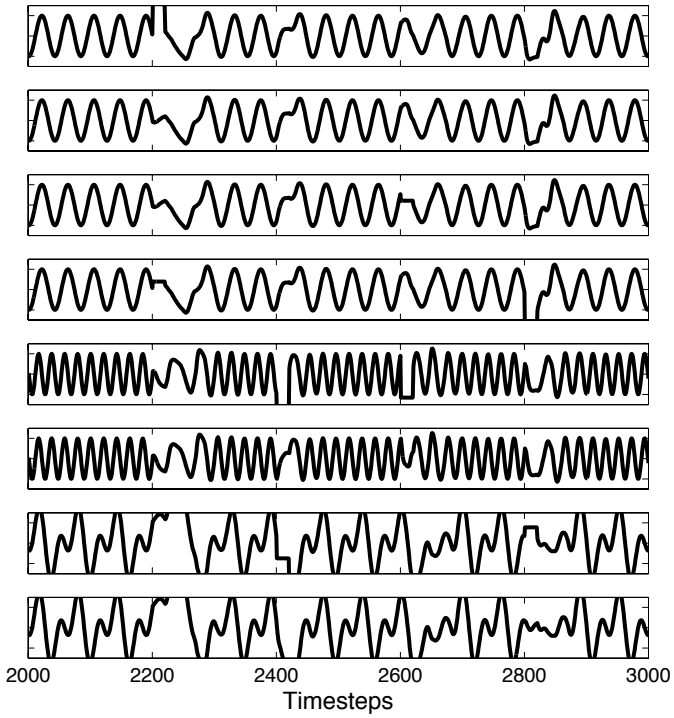


Fig. 9.3: To illustrate the capabilities of the NCPG, we trained our system using an 8 dimensional trajectory. The first four dimensions were set to $\sin(0.15k)$, the second two to $\sin(0.3k)$ and the last two to $\sin(0.2k) + cos(0.1k)$. After training the system was left freely and every 200 time steps we clamp one or two random outputs to a random outlier value. As one can see, the system is able to cope with these perturbations and due to the coupling of the reservoir all outputs remain coupled.

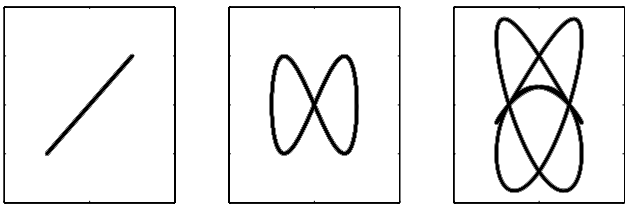


Fig. 9.4: Illustration of the output coupling of a trained NCPG. The three phase portraits show the coupling between the outputs of our system after it recovered from perturbations. The output dimension was 8, the most left graph show the coupling between the first two outputs, the middle graph shows the coupling between the first and fifth output and the right graph shows the coupling between the first and eight output. Once recovered from the perturbations, we can see that all outputs remain coupled.

Mode Both offline and online training are possible. Online training can be useful to learn to deal with sensor feedback.

9.4.5 Training Data

Usually, example trajectories (angle evolution to control the servo motors of a robot) are used as training data. If the system has to be able to switch between multiple different trajectories (e.g., different gaits), additional inputs can be defined which denote the gait presented to the system. After training, these inputs can be used to switch from one gait to another (Figure 9.2). Also, high-dimensional trajectories, as shown in Figure 9.3, can be learned by the NCPG.

9.4.6 Generalization

Generalization is not yet defined properly for periodic systems. Several interpretations are possible. We define generalization as the ability to recover from unseen conditions. As shown in Figure 9.4 random perturbations can be applied to our systems. The system is able to recover from them. Apart from that, the system is able to deal with noisy feedback which can be useful whenever the output is applied on a robot and the output feedback comes from the robot sensors. We elaborate more on this in section 9.4.8.

9.4.7 Modulation

As can be observed from Figure 9.2, both frequency and shape can be modulated. However, modulation capabilities might be increased further to other properties. The frequency can be modulated by changing the leak-rate λ of our system. Amplitude modulation can be done by amplifying the output and decreasing the output feedback in the same way. When shape changes are desired, one has to train our system showing all desired trajectories together with one or multiple inputs which denote which trajectory has been shown. Usually one additional (binary) input is taken for each shape that has to be generated. When a certain trajectory is shown (or has to be generated) the corresponding input is set to 1, otherwise the input is set to -1.

9.4.8 Sensory Feedback Integration

Sensory feedback integration can be done easily by adding additional inputs to the system or using the output feedback loop. As illustrated in Figure 9.5, FORCE learning makes it possible to apply the output of the NCPG on a system, for example to control the joints of a quadruped robot, and get feedback from the sensors of this robot (for example from the rotary encoders in all joints) during training. This enables learning relations between the NCPG output and the sensor feedback directly. This might be helpful to make locomotion control using a NCPG robust against perturbations applied on the robot.

9.4.9 State Variables

The number of states is limited by computational effort only and is defined by the number of neurons. Typically, we use large recurrent neural networks of more than 100 neurons which can be increased up to more than 1000 depending on the complexity of the task (number of gaits, output dimensionality,...).

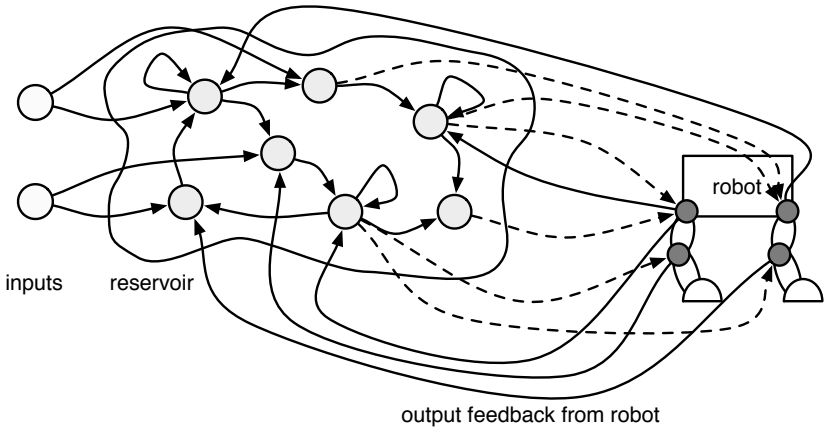


Fig. 9.5: Illustration of the ability to add sensory feedback. Since our system allows in loop feedback during training, the system can learn the relation between the output of the NCPG (the applied control signals) and sensor feedback. As suggested in this sketch, the NCPG generates the control signals for each joint of the robot. The feedback is derived from the rotary sensors in each joint. Additionally, some other sensors or control inputs can be fed into the system. Only the reservoir-to-output connections are trained (dashed lines), all other connections are randomly created when constructing the system.

9.4.10 Robustness and Adaptation to Perturbations

In Figure 9.3 we illustrate the ability to cope with perturbations on the output feedback of the NCPG. Every 200 time steps one or two randomly chosen outputs are clamped to a random outlier value. From our experiments, it seems that the system is able to recover from noise or perturbations of all inputs and state variables. Even when all states are reset to some random value, the system can recover.

9.4.11 Stability

All neuron states are bounded by the saturating nature of the nonlinearity. Additionally, FORCE tends to keep the reservoir-to-output weights small. As a result, the output of the NCPG will be bounded for any applied input. Additionally, we learned from experiments that the NCPG converges to it's attractor, even under difficult conditions (large disturbances). However, to our knowledge no proof can be given that the NCPG will converge to its attractor under all conditions.

9.5 Non-Functional Analysis

9.5.1 Representation and Interface

The output of the NCPG is a rhythmic trajectory representing the control signals for the servo motors of a robot. Typically, angle evolution is used. The output can be high-dimensional when multiple servo motors have to be controlled. The input is mainly defined by the output feedback which can be derived directly from the system's output. Alternatively, the output feedback can come from the robot sensors (rotary encoders) which generate a delayed and/or filtered form of the system's output. Additionally, multiple inputs can be added to modulate the output of the system or

to integrate other robot sensors. It is desirable to normalize both inputs and outputs, to avoid saturation of the neuron states. The system has multiple parameters which are summarized in Table 9.1. The leak-rate can be adjusted at runtime in order to modulate the output frequency.

9.5.2 Timing

Time is determined by the iteration step size. During testing, computation time depends on equation 9.1 and 9.2.

9.5.3 Robustness and Reliability

The NCPG's response time is predictable and does not depend on its input variables. It will respond to queries at all time during execution. Computation time depends on equation 9.1 and 9.2 and meeting real-time constraints shouldn't be any problem if the reservoir is not too large.

9.5.4 Dependencies

Since the NCPG can be applied in open loop, the system is not dependent to any exogenous inputs. Proprioception can be added to the NCPG in two ways: by using sensor information for the output feedback and by adding sensor information as additional inputs to the system. Currently we are using Python to implement the system using the Scipy library. The system can be provided in matlab as well.

9.5.5 Runtime

Different stages can be identified for our system: offline learning, online learning, evaluation (testing) and recognition (see Section 9.5.6). Both offline and online learning occur step by step using equations 9.1 to 9.5. When applying online learning, the output can be applied on the robot directly or the feedback can be derived from the robot sensors. For testing, the reservoir-to-output weights $\mathbf{W}_{\text{out}}^{\text{res}}$ are kept fixed and only equation 9.1 and 9.2 have to be applied every time step.

9.5.6 Usefulness for Recognition

If the NCPG is trained such that the inputs that are responsible for gait selection are considered as outputs that have to be predicted (apart from the desired trajectories), output weights will be trained for both inputs and outputs. In this kind of setup, one can chose whether the NCPG has to generate a desired trajectory by applying the correct input, or that the correct input (which acts as a classification output now) must be predicted by feeding a trajectory to the NCPG. This makes the NCPG suitable for gait classification.

9.6 Summary

In this chapter we have presented the Neural Central Pattern Generator (NCPG). The core of the system is a random recurrent neural network. Training is done by using FORCE learning. This means that Recursive Least Squares is used to train (only) the output weights while the real output of the system is fed back into the reservoir. The NCPG is capable of learning complex, high-dimensional rhythmic patterns (e.g. control signals used for locomotion) and to stably generate them. Simulations show that the system is robust against perturbations. The NCPG output frequency can be

modulated and if desired several gaits can be trained such that the system can switch at runtime from one gait to another.

Chapter 10

Comparison

In this chapter a comparison between models described in previous chapters is presented. The analysis and comparison regarding *Reaching Tasks* and *Periodic Tasks* models are separated in two different sections. Then an overall discussion is presented to analyze the suitability of different models for the complete architecture that will be developed within the AMARSi project.

To have a comparative study, it is necessary to analyze the models from different perspectives. Collecting the information presented in previous chapters creates table 10.1. Columns in table 10.1 refer to the different models while each row describes a certain aspect. The key features of this table and previous chapters are presented in the following sections.

10.1 Models for Reaching Tasks

There are six models that are considered to be used for reaching tasks:

1. Dynamical Movement Primitives (DMP): Considering an analytically well understood dynamical system with good stability properties and modulating it by nonlinear forcing term to achieve a desired attractor behavior.
2. Stable Estimator of Dynamical Systems (SEDS): Estimating multidimensional autonomous nonlinear dynamical systems based on optimizing the model for demonstrated motions under the constraint of the model's global asymptotic stability.
3. Neural Dynamic Movement Primitives (NDMP): Coupling task and joint spaces by means of an attractor-based reservoir network and associative learning. Feedback from the robot is in the loop of learning.
4. Neural Dynamical Motion Primitive Generator (NDMPG): Using a recurrent neural network to code the desired reaching movement in an attractor landscape, based on the echo-state approach.
5. Neural Motor Primitive Control (NMPC): Controlling (closed loop) a dynamical system that models the dynamics of a robot based on a recurrent neural network.
6. Planned Motion Primitive (PMP): A local optimal trajectory planner based on a subgoal representation using a state-of-the-art probabilistic planning method.

These models are analyzed in the following subsections.

Tab. 10.1: The complete table of different adaptive modules and their properties. Models for both reaching and periodic tasks are present.
The complete names of acronyms used in the "Learning algorithm" row are:
LWR: Loclly Weighted Regression, SEDS: Stable Estimator of Dynamical Systems, LREG: Linear REGression, BPDC: Back-Propagation De-Correlation, CMA-ES: Covariance Matrix Adaptation Evolution Strategy, LAWER: Locally-Advantage Weighted Regression, FORCE (not an acronym): FORCE-learning is a procedure for modifying internal or external connection weights to change the chaotic spontaneous activity of a recurrent neural network.

	DMP	SEDS	NDMPG	NMPC	NDMP	PMP	AFO	NCPG
Dynamics	nonlinear convergent	nonlinear convergent	recurrent neural net	recurrent neural net	recurrent neural net	planned trajectory	nonlinear convergent	recurrent neural net
Nonlinearity	parametrized Gaussian functions	mixture of Gaussian functions	parametrized sigmoidal functions	parametrized sigmoidal functions	parametrized sigmoidal functions	planned trajectory	weighted sum of oscillators	parametrized sigmoidal functions
Attractor	point attractor, limit cycle attractor	point attractor	point attractor	point attractor	network point attractor	subgoals	limit cycle attractor	limit cycle attractor
Coupling	one system per DOF	coupled system	coupled system	coupled system	coupled system	coupled system	one meta-oscillator per DOF	coupled system
Training data	sample trajectories	sample trajectories	sample trajectories	none or sample trajectories	sample trajectories	sample trajectories (for learning forward models)	sample trajectory, sensory information	sample trajectories
Learning algorithm	LWR	SEDS	LREG	LREG, FORCE	BPDC, LREG, FORCE	CMA-ES (Genetic alg.), LAWER	embedded in the dynamics	LREG, FORCE
Learning type	supervised	supervised	supervised	unsupervised / supervised	supervised	supervised / reinforcement learn. (for finding subgoals)	supervised	supervised
Learning mode	online / offline	offline	offline	online / offline	online / offline	online	online / offline	online / offline
Number of state variables	4 × dimensions	2 × dimensions	network size	network size	network size	1 or 2 × dimensions	2 or 3 × pool size	network size
Number of parameters needed to be tuned	1 (number of basis functions)	1 (number of Gaussians)	1 (input scaling)	2 (input scaling, time shift δ)	4 (step size of IP + BPDC, two regularization parameters)	1 (Number of subgoals)	1 (pool size)	2 (leak-rate, input scaling)
Sensory Feedback Integration	manually defined	from state variables and user pre-defined desired variables	manually defined	learned by example	proprioception	manually defined	manually defined	learned by example
Asymptotic convergence to the desired behavior after perturbation	global-proved	global-proved	local-empirical	local-empirical	local-empirical	local-proof dependent on kinematics model	global-proved	local-empirical
BIBO Stable	yes	yes	yes	yes	yes	yes	yes	yes

10.1.1 Coupling and Dimensions

Coupling is an important timing issue needed for synchronous performance. If two tasks are completely decoupled, they can easily get out of sync with even small perturbations. Coupling between different dimensions might have two different meaning: unilateral and bilateral couplings, i.e. one process affecting another (but not in return), and two processes affecting each other mutually. From the models presented, a single 1-DOF DMP is decoupled. When the canonical system is shared between different dimensions (1-DOF subsystems), the whole model is unilaterally coupled through the canonical system (transformation systems are not coupled directly). Also, DMPs can have multiple canonical systems that are bilaterally coupled. SEDS, NMPC, NMPG, NDMP and PMP are bilaterally coupled multi-dimensional systems. The models are N-dimensional and they model all of the dynamics in a single model.

To summarize, DMP has different transformation systems for different dimensions and SEDS, NMPC, NDMPG, NDMP and PMP model all the dimensions in one single model.

10.1.2 Need for Kinematics Learning

If a good definition for inverse kinematics of the robot is unknown, using a model that learns the kinematics is beneficial. Among the models introduced, NMPC and NDMP are capable of learning inverse kinematics. They both use sets of N-tuples {end effector position(t), joint angles(t)} to learn the inverse kinematics. However, they are not designed to learn raw input trajectories. This is where DMP, SEDS and NDMPG are useful. They can learn input trajectories and code them into dynamical systems, without depending on the meaning of the given dataset. On a higher level, PMP just does optimal trajectory planning and do not store the trajectory in the dynamics. PMP assumes a given forward kinematic model, which can be learned by any supervised learning method like NMPC or NDMP. Then, it uses optimal control to plan a low cost trajectory.

So, here, the issue is to define the need. If the desired trajectory is not defined, or if it is only defined for the end effector, knowing (learning) robot's kinematics is a necessity to execute commands on the robot. So, if it is required learn the inverse kinematics, NMPC and NDMP are the choices, but if the adaptive module is meant to learn a raw input trajectories, DMP, SEDS and NDMPG have to be used. Finally, PMP is meant to be used in high level as the optimal control planner and do not encode any trajectory itself. So, PMP is useful when a well defined trajectory is not present, and it is needed to optimally follow some subgoals.

10.1.3 Parameter Modulation

Considering the models for reaching tasks, the first important modulation ability is the goal modulation, i.e. the ability to create movements towards new goals that were not part of the training data. DMP is capable of doing goal modulation through the g parameter. It is important to mention that the outcome of the goal modulation is dependent on the choice of the coordinate system. Additionally, the behavior is different if goal modulation is done in the beginning or during the movement. SEDS defines the goal in the origin of the coordinate frame. SEDS does the goal modulation through moving the origin of the coordinate frame. NMPC changes the goal position with the definition of ξ_{des}. NDMPG's goal is always at the origin of the coordinate system, so goal modulation is done by correct offsetting translation of the coordinate system. For NDMP, in the autonomous reaching mode, goal modulation is done by changing the desired input values. Finally, goal modulation for PMP can be done trivially by changing the last subgoal. However, re-planning is needed in this case.

DMP and SEDS are globally asymptotically stable, so changing the goal to arbitrary places is possible for them. However, it is notable that the way SEDS and DMP modify the motion to reach the new goal is different. Since DMP learns a motion from one single demonstration, the generalizations may not necessarily result in a desired behavior, for a large change in the goal. DMP just scales the path to reach the goal when the goal is modulated. SEDS is able to learn the basin of attraction to the goal from any number of demonstrations. Hence it can generalize the motion more properly in different areas in space.

The modulation of the goal state for NMPC, NDMPG and NDMP is also dependent on the training data. If these models are well trained, they can modulate the goal in the neighborhood of the training data. But if the desired goal is outside of the covered area, NMPC and NDMP cannot guarantee reaching the goal. For NDMPG, since the network is damped, it will always (empirically) go to 0, which means the goal position will be reached eventually. It is needed to mention that NDMP is designed to do an additional goal modulation task: when the goal state is incompletely given, and auto-association is needed. This is when only one part of the desired attractor state is given, the other part shall be associated (e.g. desired end effector position in the goal state is known, but corresponding posture is not known). For PMP, trajectory is represented by a sequence of subgoals, representing a higher level plan. This subgoals can be changed to an arbitrary position as long as the kinematics planner is able to handle that. So, the reliability is dependent on the kinematics model.

Second, it is important to analyze the generalization to the initial state. For DMP, since it assumes that the initial state (y_0) is zero, change in the initial state is interpreted as a translation in the goal state. SEDS and PMP has the current state (position) as the input, so change in the initial position is easily accepted. This is the same for NMPC, NDMPG and NDMP, with the difference that convergence to the desired goal is not analytically proven. It holds for all of the models that having a desired behavior (from different initial conditions) is partially dependent to the coverage of training data.

Third, it is important to know which models can encode the desired trajectories in themselves. DMP, SEDS and NDMPG can code arbitrary reaching trajectories and accept them as training data. NMPC is also able to regenerate the trajectories used for inverse kinematics learning (if any) by the transients of RC-networks (but the regeneration of similar trajectories is not as good as DMP, SEDS and NDMPG). For NDMP however, learning movement shapes (trajectories) is still ongoing research. Additionally, SEDS, NMPC and NDMPG can code multiple trajectories while DMP can only code a single trajectory at once. So SEDS, NMPC and NDMPG are capable of doing a reaching task in different ways while DMP needs re-learning, or using multiple models and switch between them, to do a task in a new way. PMP does not store the desired trajectory (subgoals) in the dynamics and only uses it as the input for planning.

Fourth, it is an advantage for an adaptive module to adapt to the changes in the robot/environment. Here, we do not mean change in robot states, like configuration or end effector position, but we mean something like change in a link length, or a changing environment. While NMPC tries to automatically adapt itself to the change in environment and it explores the environment to learn the kinematics, DMP and NDMP require the appropriate response to be defined so they can adapt through online learning. For SEDS, if it is working in Cartesian coordinates (end effector control), and if the change is affecting the kinematics model used, it is up to the kinematics model to adapt. But if the change in the environment can be perceived by SEDS, like change in an obstacle's position, it can adapt well. This is the same for NDMPG. NDMPG has an extra input to define the position of an obstacle. At last, the behavior of PMP depends totally on the kinematics learning model and cannot adapt itself separately.

Finally, all of the presented models are able to modulate the speed of performing a desired task. DMP performs this through its τ variable, SEDS and NDMPG with the modulation factor λ (output filter control for NDMPG), and NMPC and NDMP with the leak-rate parameter. Regarding PMP, smaller control weights and the timing of the subgoals can be used to modulate the movement execution time.

10.1.4 Learning

Learning can be described in different manners:

- online vs. offline
- supervised vs. unsupervised vs. reinforcement learning
- one-shot vs. iterative

Starting with DMP, it is capable of both online and offline learning in a supervised manner with locally weighted regression (LWR). There are also techniques of reinforcement learning (to be exact, policy search) used to train DMPs [2]. Essentially, those techniques can be extended for other parameterized dynamical models with online learning. SEDS is designed to use supervised offline learning as it trains the dynamical model through constrained nonlinear optimization. NMPC provides both online and offline supervised learning with recursive least-squares, where NDMPG only implements supervised offline learning through ridge regression. For NDMP, there are two learning mechanisms: 1) unsupervised reservoir optimization with intrinsic plasticity to improve the encoding capabilities, and 2) supervised read-out learning using either online backpropagation-decorrelation or offline ridge regression to learn the kinematics. Intrinsic plasticity is applied in parallel with backpropagation-decorrelation learning in the online learning scenario. In the offline case, intrinsic plasticity is applied in a pretraining phase before read-out regression. As for PMP, learning takes place at two stages. The kinematics can be learned by any supervised learning technique like NDMP or NMPC. Whereas the optimal subgoals are computed iteratively by the use of reinforcement learning methods. This optimization can be done online or offline (PMP is not included in the further discussion because of the basic differences).

Learning for DMP in the offline mode is one-shot. DMP in the online mode, SEDS, NMPC, NDMPG, NDMP and PMP need time to converge to the minima. The computational time needed for convergence is dependent on the size of the model, the initial values and the complexity of the learning algorithm.

When the learning method is defined, one has to define what are the properties of the data collection for training. DMP, uses a trajectory-set containing triplets of an entity and its first and second derivatives (e.g. position, velocity and acceleration). SEDS is flexible in terms of input and output parameters, and it can accept any arbitrary variable for training. For example, the training data-set for SEDS could be any combination of position, joint angles, end-effector orientation, velocity, acceleration, force, torque, etc. NDMPG uses end effector trajectories for training. NMPC can be trained without any training data and uses internal exploration to build up the model (but it seems to need a good initial estimate). Nevertheless, it can be trained with {end effector, joint angles} pairs, which is the same for NDMP.

Another important issue is that how many trajectories are needed for the models to be trained. DMP is trained with a single trajectory. If there are multiple learning trajectories, the outcome is an average of them. So, the training set should be for a single task. SEDS accepts multiple demonstrations, which can even have inconsistencies (having different behaviors in same states). It can also be trained by a single demonstration, but the generalization is better with multiple demos. NMPC can learn the inverse kinematics without any training data. However, if the internal model is meant to be trained with examples, a large amount of training data is recommended,

alike for NDMPG. Finally, NDMP can easily handle multiple trajectories – in particular as the basic version does not store their geometrical form. NDMP needs very few samples. Kinematics of 7 DOF can be learned with only 250 data points for one workspace.

10.1.5 Time Complexity and Integration

One key comparison between different adaptive modules is their time complexity in the evaluation phase, or, in other words, computational load to evaluate a single N-dimensional entry. This entry can be end effector position, joint angles, etc. What is important is the dimensions of the entry (N), and analysis will be a function of N. Time complexity analysis is very important since it defines the limits for the real-time design.

The basic operation to analyze the time complexity is assumed to be the **multiplication of two scalars**. Additionally, since the nonlinearity of all of the models discussed here are originated from a sigmoidal or Gaussian function, without loosing anything, it is considered that **evaluating a 1-D sigmoidal or Gaussian function with a single data point** is a basic operation. With these assumptions, a coupled N-dimensional DMP will be of $O(K \times N)$ where K is the number of basis functions in each dimension. SEDS is of $O(K \times N^2)$ where K is the number of Gaussian functions. For NMPC, the time complexity is of $O(P^2 + P \times i + P \times o)$ where P, i and o stand for pool size, input and output size. Assuming that the number of neurons are more than inputs and outputs, time complexity can be rewritten as $O(P^2)$. This the same for the NDMP in evaluating the transients of the network. For NDMP evaluating in association mode, the time complexity is of $O(M \times P^2)$ where M is the number of iterations until convergence. If a trajectory is discretized into S discrete time steps, then the time complexity of the PMP is given by $O(S \times N^2 \times M)$, where N denotes the dimensions of the system and the number of planning iterations is indicated by M.

Another important question is the sensitivity of the models to the integration timestep and method. It can be assumed that, for simplicity and efficiency, an Euler integration method is going to be used. It is clear for DMP, SEDS and PMP that they can work even with fairly large timesteps. As for NMPC, as there is not any guaranteed convergence margin around the designed trajectory, the model is probably more sensitive to the choice of timestep. This sensitivity is even more tangible for NDMP where the trajectory to reach the goal is not defined. It is important to mention that NMPC and NDMP are discrete dynamical systems and do not need any integration method.

10.1.6 Sensory Feedback Integration

Sensory feedback integration helps a model to respond properly to different situations, e.g. due to changes in the environment and the robot. Sensory feedback can take different modalities, ranging from proprioception to sensory information like gyroscopes, accelerometers, etc. DMP accepts sensory feedback as additive coupling terms. The coupled term is a function of the sensory information, and the behavior of this function has to be defined manually. NDMPG is an end effector trajectory generating system that accepts sensory feedback through input slots. SEDS accepts proprioception (joint angles or end effector position depending on the training data content) as an input, so this system can work in a closed-loop. In addition to proprioception, SEDS can also accepts sensory information through additive coupling terms through a phase variable. NMPC and NDMP are more advanced in regards of incorporating sensory feedback. They both have proprioception as inputs, so they work in a closed-loop. Additionally, they can accept other sensory information as additional inputs. The added

input will actually affect their behavior. Especially for NMPC, since it has its own learning-through-exploration, it can find the function to be imposed on the sensory information. PMP uses proprioception within the feedback controller. An integration of additional sensory feedback is difficult, because an exact model of the sensors and the environment would be necessary. However, if the information is available it can be integrated into the cost function.

10.1.7 Behavior After Perturbations

Perturbations can happen as spatial and/or temporal perturbations. A spatial perturbation means that the value of a DOF is changed by an external source. But, a temporal perturbation is happening for instance when a DOF is frozen for a timespan, so the value of that DOF cannot change. Both spatial and temporal perturbations can happen for multiple dimensions simultaneously.

DMP is able to handle spatial perturbations of any amplitude since it is globally stable. For a DMP working in open-loop, the reaction to a spatial perturbation is trying to reach the desired trajectory while the y state is evolving by time. The reaction of DMP to a temporal perturbation is the same as to a spatial perturbation. The model tries to reach the trajectory toward reaching the evolving y state. This is not exactly the desired behavior for a temporal perturbation because it will go off-track and skip a part of trajectory to reach the evolving state. A good idea is to have different feedback functions to define the desired behavior in different situations, and this topic needs more research.

SEDS have advantages in handling temporal perturbation. While SEDS can handle spatial perturbation with any amplitude, it can also handle temporal perturbations without going off-track. As another advantage, since SEDS is generally trained with multiple demonstrations, it will not necessarily try to go back to the trajectory before perturbation, and it selects a new way (based on training demonstrations) to reach the goal.

NDMPG woks in closed-loop in Cartesian coordinates (end effector position), so reacting to spatial perturbation is on-the-fly. After a spatial perturbation, NDMPG goes back to the trajectory before perturbation, and goes from another track to the goal point. However, it is possible for NDMPG to get stuck in an undesired attractor state if the perturbation is large. For temporal perturbation, since NDMPG perceives current end effector position, it will pause until the temporal perturbation is over, and then continue from the point before perturbation. For NDMPG, reaching the goal is not proven, but it is empirically shown.

NMPC and NDMP handle spatial and temporal perturbations in a same way. A perturbation is a change in the input that is correspondent to proprioception. The response to perturbation is dependent on the amplitude of perturbation, learning data provided and the generalization capability of the trained model. Based on these issues, the response to a perturbation can be good or bad, and nothing is guaranteed. Nevertheless, NMPC and NDMP seem to converge back to the desired trajectory if the perturbation is not large.

PMP can theoretically react optimally to arbitrary (and limited) Gaussian noise present in the system dynamics (that is not forcing the system outside the belief of the planned trajectory). This is investigated further. But an additional control mechanism would be necessary to achieve the desired control behavior in the presence of temporal perturbations. Generally, unexpected perturbations, which require a re-planning of the subgoal parameters, need to be detected by a separate module.

10.1.8 Stability

Stability is an essential aspect needed for an adaptive module to be reliable. The notion of stability has to be addressed for three issues:

1. BIBO stability: BIBO stands for Bounded-Input Bounded-Output. If a system is BIBO stable, then the output will be bounded for every input to the system that is bounded.
2. Goal's asymptotic stability: If all solutions of the dynamical system that start out near an equilibrium point g (here the goal point) stay near g forever, then g is Lyapunov stable. More strongly, if g is Lyapunov stable and all solutions that start out near g converge to g, then g is asymptotically stable. This stability can be addressed for just parts of the state space (local), or for the entire state space (global).

Based on the definitions above, all presented models are BIBO stable. It means that for bounded inputs, they will not diverge to large and increasing values. Following is the discussion for goal's asymptotic stability.

The desired goal in DMP is inherently asymptotically globally stable. The main idea of DMP is to use a globally stable dynamical system and shape its behavior without loosing stability. DMP's asymptotic stability can be analytically proven using contraction theory. SEDS is also asymptotically globally stable. The stability is guaranteed through the constraints put in the learning algorithm.

PMP itself is locally stable around the posterior belief of the planned trajectory. However, this stability is dependent to the lower level kinematics model that is used.

On the other side, NMPC, NDMPG and NDMP do not have any analytical proof of asymptotic stability. They all have big pools of neurons that are the source of the dynamics, and finding constraints and proofs for asymptotic stability is a very complicated task. Nevertheless, numerical tests show that NMPC, NDMPG and NDMP can be reliably asymptoticly stable in the locality of the training trajectories. This is due to the fact that dynamic reservoirs with damped weights work like a damped-spring.

10.2 Models for Periodic Tasks

Based on the models provided, three of them are designed to learn and perform periodic tasks:

1. Dynamical Movement Primitives (DMP) - periodic version: Considering an analytically well understood dynamical system with good stability properties and modulating it by nonlinear forcing term to achieve a desired attractor behavior.
2. Adaptive Frequency Oscillators (AFO): Extending standard CPGs to allow the system to learn the frequency of a periodic input signal, and adapt its own intrinsic frequency to it.
3. Neural Central Pattern Generators (NCPG): Using a recurrent neural network, the reservoir, in a chaotic state to autonomously generate rhythmic patterns.

A comparison between these models is presented in the following subsections.

10.2.1 Coupling and Dimensions

DMP defines separate subsystems for different DOFs. Then, a shared canonical system is used to unilaterally couple them (it is also possible to have multiple canonical systems). The canonical system plays the role of central clock for this system, so the whole system is always coupled.

A single AFO is a decoupled part that is able to adapt its intrinsic frequency to one of the frequencies in the input signal. To acquire a desired signal shape, even for one

DOF, a pool of coupled AFOs is used. The coupling is done unilaterally with the first AFO in the pool, but it can be simply done bilaterally. So, if there are K frequencies in each dimension, total number of coupled AFOs to learn an N-dimensional input is $K \times N$.

While the elements of DMPs and AFO are decoupled, NCPG is a completely coupled system. NCPG tries to learn and code all of the dynamical behavior in a single reservoir. NCPG does not have any internal clock or phase definition.

10.2.2 Parameter Modulation

The first parameter that is of importance is the frequency. Luckily, all of the three models are able to do frequency modulation. In DMP, AFO, and NCPG goal modulation is done by changing the τ, ω and leak-rate λ variables respectively. While changing the frequency parameters in DMP and AFO is straightforward, to change the frequency of the NCPG to a desired one, it is needed to calculate the proper leak-rate(λ) value.

Another parameter that can be modulated is the amplitude of the generated signal. DMP does this through changing the explicit r variable. AFO need to change the internal radius variable of the oscillator, e.g. μ for Hopf oscillator. For DMP and AFO, amplitude of each dimension can easily be modulated to a different value without loosing coupling. If a NCPG is meant to do the amplitude modulation, training data must include the desired behavior with different amplitudes. As an alternative, one can upscale the output and downscale the output feedback by the desired factor.

Modulating the shape of the generated signal is one of the capabilities of NCPG. NCPG can accept different trajectories as training inputs and manually switch between them in the evaluation phase. This is not possible for DMP and AFO (pool of them albeit) since they can only code a single behavior at once. However, they can change the shape of the generated signal implicitly (blindly). DMP can do this with changing the w variables in the forcing term, that is proportionally related to the shape of trajectory. AFO can also change the frequencies of different nodes, but guessing the proper change is not straightforward at all.

Another feature that can improve the generalization capabilities of a rhythmic adaptive module is the ability to modulate the speed of convergence to the limit cycle (in other words the duration of the transient behavior), without changing the frequency. For DMP, this can be achieved by changing the α_z and β_z values (spring and damper properties). For AFO, this is achieved if the oscillators used have such parameter. The γ parameter is defined for this reason for the Hopf oscillator. For current state of the NCPG, this ability is not yet defined.

Finally, it is important to know if an adaptive module is capable of changing phase lags between different dimensions online. One use of this feature is changing the gait (in locomotion) based on changing phase differences between joints. For DMP and AFO, this can be done by changing the coupling matrix between different dimensions. To do this, DMP needs having multiple canonical systems, that are coupled. For a NCPG, changing the phase differences between dimensions is dependent on having proper training data (covering different phase-lag choices) and a switch (input) to change between them online.

10.2.3 Learning

All of the three models, DMP, AFO and NCPG, use supervised learning. They can do the learning procedure in both offline and online modes. DMP uses locally weighted regression, that will always end in a good estimation of the trajectory if the number of basis functions is not too low. A pool of AFOs tries to find the frequencies of teaching signal, but based on the initial values of the intrinsic frequencies, the pool size, and

the coupling strength, learning may end in with a perfect result, or a not-so-good one. For a NCPG, the risk of the system to be bad depends more on the properties of the reservoir and not of the training itself. Training is done by applying linear regression (or recursive linear regression), and thus, if the reservoir is constructed properly, a good model is obtained. However - with all these parameters - one could have the problem of over-fitting, but this can be solved by applying a good regularization technique (ridge regression, FORCE learning, etc).

10.2.4 Time Complexity and Integration

To analyze the time complexity of the models, first one needs to define the basic operations. Here, it is assumed that the **multiplication of two scalars** is a basic operation. Moreover, without loosing anything, it is considered that **evaluating a 1-D sigmoidal or Gaussian function with a single data point** is a basic operation. It is important to mention that the time complexity analysis is derived for execution in the evaluation phase (not the learning phase).

Time complexity of a coupled N-dimensional DMP will be of $O(K \times N)$ where K is the number of basis functions in each dimension. A pool of AFOs is of $O(K)$ where K is the number of AFOs in the pool. For NCPG, the time complexity is of $O(P^2 + P \times i + P \times o)$ where P, i and o stand for pool(reservoir), input and output size. Assuming that the number of neurons are much more than inputs and outputs, time complexity can be rewritten as $O(P^2)$.

Another important question is the sensitivity of the models to the integration timestep and method. It can be assumed that, for simplicity and efficiency, an Euler integration method is going to be used. It is clear for DMP and AFO that they can work even with fairly large timesteps. NCPG do not even use an integration timestep since it is discrete dynamical system. Concluding, the integration timestep is not a serious concern for the presented periodic adaptive modules.

10.2.5 Sensory Feedback Integration

Sensory feedback integration helps a model to respond properly to different situations of robot and environment. Sensory feedback can be different things, ranging from proprioception to sensory information like gyroscope, accelerometer, etc. DMP and AFO accept sensory feedback as additive coupling terms. The coupled term is a function of the sensory information, and the behavior of this function has to be defined manually. NCPG accepts sensory feedback as inputs to the model. As NCPG will take into account whatever input it is given, regardless of its physical meaning, the response to a sensory feedback depends on a proper training. Otherwise, the response will be emergent, and not necessarily good.

10.2.6 Behavior After Perturbations

Perturbations can happen as spatial and/or temporal perturbations. A spatial perturbation means that the value of a DOF is changed by an external source. But, a temporal perturbation is happening for instance when a DOF is frozen for a timespan, so the value of that DOF cannot change. Both spatial and temporal perturbations can happen for multiple dimensions simultaneously.

DMP is able to handle spatial perturbations of any amplitude since it is globally asymptotic stable. The reaction to a spatial perturbation is trying to reach the trajectory, while the subgoal on the trajectory is evolving by time. A good idea is to have different feedback functions to define the desired behavior in different situations, and this topic needs more research. The reaction of DMP to a temporal perturbation is the same as to a spatial perturbation. The model tries to reach the trajectory toward

reaching the evolving subgoal. This is not exactly the desired behavior for a temporal perturbation because it will go off-track and skip a part of trajectory to reach the evolving subgoal.

AFO can handle spatial perturbations in the evaluation mode. This is true for both a single AFO, and a pool of them. But, for a pool of AFOs, it is hard to have a direct proprioceptive feedback. As for the temporal perturbation, with a good coupling formulation for proprioception, a stopping, speeding down or speeding up phase behavior can be achieved with a single AFO (regarding the application). This discussion is not valid for a pool of AFOs since each single x state variable is not separately linked to a physical subject (like a joint angle).

For a NCPG, since it incorporates inputs for proprioception, spatial and temporal perturbations are handled the same. A perturbation just changes the input to an unexpected value. The response to a perturbation depends on the learning data, and whether it covers the perturbed situation, and the generalization capability of the learned model. Convergence to the desired periodic behavior is not guaranteed after perturbation. Nevertheless, a good convergence behavior in locality of the training data is empirically shown.

10.2.7 Stability

Stability can be addressed as BIBO stability and asymptotic stability (see 10.1.8 for more). Stability is an essential aspect needed for an adaptive module to be reliable. All of the three models are BIBO stable.

DMP is inherently globally asymptotic stable. Asymptotic stability of AFO is dependent on the choice of the oscillator. With a Hopf oscillator used, the AFO is globally asymptotic stable. This is generalized to a pool of AFOs.

NCPG does not have any analytical proof of asymptotic stability. It has big pools of neurons that are the source of the dynamics, and finding constraints and proofs for asymptotic stability is a very complicated task. Nevertheless, practice shows that NCPG can be reliably asymptoticly stable in the locality of the training trajectories.

10.3 Architecture Point-of-View

The discussion until now has compared key properties of different approaches, but has not yet addressed whether the proposed approaches are suitable to be used as building blocks for a complete, hierarchical, architecture. Indeed for this, the adaptive modules should present several features such as the possibility to work at different time scales (e.g. short time scales at the bottom, large time scales on the top), the possibility to be coupled to other modules while keeping stability of the overall architecture, encapsulation (i.e. provide explicit inputs and outputs), meaningful phase information, etc. We address these topics first for discrete and then for reaching tasks.

10.3.1 Reaching Tasks

The first thing to be discussed is if all of the models, DMP, SEDS, NMPC, NDMPG, NDMP and PMP, are suitable to be encapsulated. By encapsulation, we mean that all inputs and outputs (and any other variable needed for analysis) are explicitly expressed, and are accessible. Fortunately, all of the presented models have explicit input and output variables that helps encapsulation. As for the observability of the models, DMP, SEDS and PMP have small numbers of state variables that makes the state of the system easily observable. For NMPC, NDMPG and NDMP, this is a little bit more complicated since they have many internal state variables, so more memory is needed to have the track of the system.

The second issue is the problem of timescales. Actually, DMP and SEDS can work in very small timescales since they are both smooth and have proved convergence. Choice of the timescale has to be more careful for NMPC, NDMP, NDMP and PMP since they can get asymptoticly unstable in large timescales (but not for small ones). With this information, one can conclude that DMP an SEDS are more suitable for real-time parts of the architecture where a very small timestep is going to be used, and NMPC, NDMPG, NDMP and PMP can work in more relaxed parts of the system.

Another issue for a system to be suitable for a complete architecture is the definition of a phase state. Here phase state is defined as a variable to tell what portion of the job has been done, and what portion is remaining. This phase state has to robust to perturbations. The phase state can be also used to coordinate the adaptive module with an external clock, if needed. DMP provides this phase state as a state variable, but it is in a logarithmic manner. To be coupled with a linear clock, or other adaptive modules, the logarithmic phase variable of the DMP has to be converted to a linearly changing value. Nevertheless, this can be done simply. SEDS, NMPC, NDMPG, NDMP do not inherit any explicit internal phase variable. For these models, in order to be used in the whole interconnected architecture, it is needed to devise some method to extract a phase definition. This can be done, for example, by internal integration of these models. Alternatively, a phase variable can be added to the training data since these models can accept any arbitrary variable as input. This phase state can be used to tell what portion of the job has been done, and what portion is remaining. Anyway, this is an issue that has to be solved by each model separately, and there might be different approaches to tackle this. As for PMP, since timing of the subgoals is an external input, a phase variable have to be defined externally. Nevertheless, PMP has an internal time variable, determining how much time is left to reach a subgoal. This time variable can be replaced with a phase variable (for partial coordination), but this needs investigation.

One interesting matter that helps drawing out a decision is the nature of the candidates. DMP can learn fast and execute fast. SEDS codes the desired behavior in a very stable and robust manner. SEDS learning is not very fast, but it can execute fast. PMP can optimally track subgoals. NDMPG is a reservoir computing model that is designed to code raw trajectories. NMPC is able to learn the inverse kinematics by exploration, and NDMP is able to code forward and inverse kinematics bidirectionally in a single structure. So, what seems to be useful is a hybrid strategy like the following (as an example): *NMPC can be used in a low level to explore the kinematics. In sequence, DMP and NDMPG can use the kinematics information with the help of a planner, to do rapid prototyping and testing. Meanwhile, NDMP uses the data explored by NMPC to build up a good kinematics model. In a longer timescale, SEDS works as a long term memory and consolidates good behaviors. Finally, PMP can be used if the desired trajectory is expressed as subgoals.* It should be mentioned that this is just an option which will be explored further in the coming months.

10.3.2 Periodic Tasks

Discussions here in this subsection will have almost same topics as the previous (10.3.1). So, the first thing to be discussed is the encapsulation. Again, fortunately, DMP, AFO and NCPG have explicit inputs and outputs and they can be easily encapsulated. Keeping track of the system needs observing the states of the models. This is simple and low cost for DMP and a single AFO, since they have small number of states. But for a pool of AFOs, or NCPG, since the number of state variables is large, more memory is needed.

Second, a phase variable is needed for coordination with external sources in the whole architecture (e.g. other adaptive modules). DMP and AFO both have explicit well-defined phase variables, where NCPG do not have an explicit phase definition yet,

but it might be possible to introduce such phase variable in the training data.

NCPG have more potential to take the lead on coding sensory feedback. NCPG have a vast structure that gives space to learn the effect of sensory feedback. For DMP and AFO, the formulation of sensory feedback has to be defined manually.

What seems to be a good decision is to use each model regarding its expertise. AFO is expert with adaptation capabilities, DMP is stable and low-cost, and NCPG is a basin of emergence. So, one choice is to have a cooperation between these three approaches. As an example, *AFO can be used to extract the resonant frequency of the robot in interaction with environment, and give this information to NCPG and DMP. Meanwhile, NCPG helps finding a good sensory feedback function. Additionally, NCPG can code all information in a coupled model, and learn the link between different dimensions inherently. Finally, DMP can play the role of encoding the desired behavior in a stable way, and will be a candidate for fast execution, testing and modulating the desired task.* This is just an option which will be explored further in the coming months. It is important to know that another good choice is to design a new adaptive module that combines good features from each model, and this should be investigated further.

10.3.3 Additional Points

There are additional points about the suitability of different models to used as the adaptive module in the AMARSi project. These points can be listed as follows:

- Very likely, adaptive modules should also have "interrupts", i.e. modules should be able to tell how well they are doing and to provide a signal when something goes wrong. So far, none of the modules have this feature. It is important to keep in mind that devising such feature is not routine, but it is important to have it.
- Learning inside a module versus learning in the whole architecture should be addressed. This will be related to WP5 and WP6 where learning paradigms and architecture issues are going to be analyzed.
- It is a question when to add new modules and how to couple them with the existing ones. This study includes recognizing new inputs, and having a mechanism to find out if a new module should be created, or the existing ones should be updated. It also includes the analysis of stability (probably using the contraction theory), especially when the new module is coupled to the other ones in the architecture.

A thorough analysis can be presented in this document, but since these point are strongly related to the other AMARSi Work-Packages, a discussion is needed.

It is important to state that none of the presented models are finalized (see the extension appendix). Good features of each model can be explored for the other ones, and the AMARSi's strategy is definitely to strengthen each adaptive module model. It is possible to have mixed adaptive modules combining good properties of different approaches.

10.4 Summary

In this chapter an extended comparison study between different models of adaptive modules was presented. Models were first studied in different aspect including coupling, learning, modulation, kinematics learning, sensory feedback integration, and stability. After that, the suitability of different models to be used in a hierarchical architecture in AMARSi was discussed. The main conclusion that comes out of this detailed analysis is that currently there is not a single approach that clearly outperforms

the other approaches and exhibits all the desired features needed for the architecture. We therefore propose for the rest of the project to investigate two main directions: (1) the creation of a new type of module that combines interesting features of the different approaches developed so far, and (2) the development an architecture that is hybrid i.e. that combines different types of modules for different functionalities.

Appendices

Appendix A

Extensions and Future Works

There are several models presented as adaptive modules (see Chapters 2-9). The presented models have several capabilities, but all of them can be improved further. Here, possible extensions and future works for each model are discussed.

A.1 DMP

The first future study on DMP is the effect of including proprioception feedback directly into the transformation system. This means that DMP will be studied in a closed-loop manner. This study will help to improve the behavior of DMP after perturbations, especially temporal ones.

Second, there is an idea of mixing DMP with what has been studied done under the topic of muscle synergies. This will help to better understand the relation of muscle activities, a minimum representation of them (through PCA, etc), and motor primitives in the joint level. It will be an extension for DMP to learn weighted basis functions to model synergies, and then having higher level weights to express tasks as weighted combination of synergies. Like this, switching between different tasks can be achieved by just switching the higher level weights.

Finally, for periodic tasks, DMP depend on knowing the frequency of the teaching signal. A continued research has been followed to mix DMP and AFO to eliminate this dependence [55], and this topic is under progress.

A.2 SEDS

Currently we have extended SEDS to a more generic framework that is able to perform discrete motions with non-zero velocity at the target (this work has not published yet). This extension allows to learn a considerably wider set of motions ranging from pick-and-place movements to agile robot tasks that require reaching/hitting a static/moving target with a specific speed and direction. It also allows the generation of more complex motions by enabling a user to define via-point(s) on the path to the final goal (an example of this requirement could be catching an object in mid-air and then placing it on a table). Another ongoing extension to our work is to endow SEDS with the ability to perform real-time obstacle avoidance while retaining all the existing favorable properties of SEDS. Hence, the resulting adaptive module make it possible for a robot to interact safely and robustly in an unstructured and dynamic environment. Further,

we are also working on formulating the periodic version of SEDS, where we model cyclic motions with a non-linear autonomous Dynamical Systems. The periodic SEDS will inherit all the current properties of discrete SEDS. Finally, SEDS is currently an offline learning algorithm which learns the parameters of a dynamical system through constrained nonlinear optimization. An ongoing research is to design the online learning version of SEDS where the algorithm forms the motion dynamics and optimizes the parameters of the model along time as the robot explores the working space.

A.3 NDMP

In the current form, the NDMP framework (see Fig. 6.1) has the ability to generate straight movements but not complex shaped trajectories without an external planner. We therefore introduce an extension to the current framework. The idea is to learn the kinematics of the robot first to get the ability to approach the goal in a straight line. Afterwards, we identify frequently used trajectories, which are used to learn a velocity mapping. In the long run, this approach aims at consolidation of frequently used trajectories in the network which then can be reproduced without external planning.

The new framework is shown in Fig. A.1 displaying the additional components for the extension above the horizontal line. The difference to the basic NDMP is that another output is added to the reservoir which is trained to generate the next target position via an integration step (control loop shown in black lines in Fig. A.1). Note that the input information is the same as it is for the original kinematic framework. The extension introduces the ability to learn movement shapes in the end effector space by generating a velocity $\dot{\mathbf{u}}$ for each position $\mathbf{u}$, i.e. representing a vector field. We use the velocities to compute the new target position according to $\mathbf{u}(k) = \mathbf{u}(k-1) + \alpha\dot{\mathbf{u}}(k-1)$. The extended loop is only used if specific movement shapes are inquired.

The velocity mapping is learned from example trajectories following the previously presented learning methods (see chapter 6). The training data for the velocity mapping is derived from the raw input trajectories by calculating the velocities at each position along the path yielding pairs $(\mathbf{u}(k), \dot{\mathbf{u}}(k))$. During operation, the extension introduces an additional dynamic system besides the reservoir. Important for this approach is the convergence of the appended system to a target attractor (fixed-point attractors for discrete movements, or cyclic attractors for periodic movements). This strongly depends on the training data. A first aid to stabilize the dynamics is to add training data that forces the end position of the trajectory to be an attractor, e.g. with the velocities given by $\dot{\mathbf{u}} = \beta(\mathbf{g} - \mathbf{x})$ for positions $\mathbf{x}$ in the vicinity of the end position $\mathbf{g}$.

In this framework, multiple movement shapes, i.e. movement primitives, are represented in several velocity mappings. That means for each primitive we need another set of read-out weights which generates the needed trajectory, but only one reservoir. Blending and mixture of primitives can then be accomplished by gating the contribution of selected velocity mappings.

A.4 PMP

Representing a motion primitive as a sequence of subgoals is a very compact representation of a movement, resulting in a low number of parameters to describe a motion. In future work we want to exploit the properties of the PMPs to acquire a large amount of movement skills.

With PMPs learning takes place at two levels of hierarchy. At the motor control level the system dynamics have to be learned. At the level of the movement primitives, good subgoals have to be found in order to achieve a given task.

Future work will focus on defining a hierarchical planner which should learn successful policies (motion primitives) for different tasks within a few episodes. On the

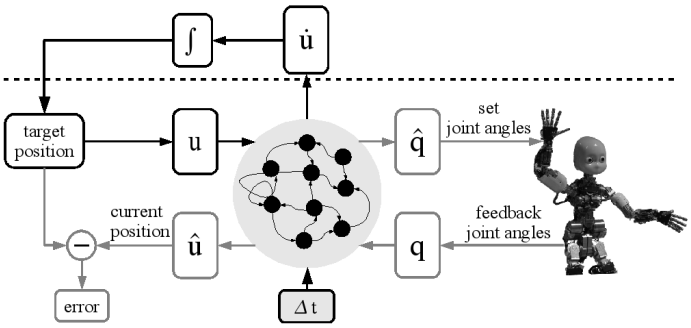


Fig. A.1: For an extension of the basic NDMP framework a velocity mapping is added (above dashed line) and used for trajectory generation (bold black components).

lowest level of the hierarchy optimal control sequences are planned, whereas learning optimal subgoals is performed on a higher level.

The probabilistic planning algorithm AICO assumes full knowledge of the system dynamics. For the evaluated balancing tasks, mathematical descriptions of the dynamic models are available. However, for more complicated robots or real world examples the dynamics are unknown or to complex to describe analytically. Learning the system dynamics is part of future work.

Finally structure learning could facilitate learning of new behavior. The probabilistic planning approach could also be used to identify and improve task relevant features, like for example the step length for walking or the center of gravity for balancing. As a result, a motion primitive would be defined by a sequence of higher order features instead of subgoals and is part of future research.

Bibliography

[1] A. J. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor, and S. Schaal, "Learning nonlinear dynamical systems models." Under review.

[2] J. Kober and J. Peters, "Policy search for motor primitives in robotics," in *NIPS* (D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, eds.), pp. 849–856, MIT Press, 2008.

[3] L. Righetti and A. J. Ijspeert, "Programmable Central Pattern Generators: an application to biped locomotion control," in *Proceedings of the 2006 IEEE International Conference on Robotics and Automation*, 2006.

[4] S. Degallier, C. Santos, L. Righetti, and A. Ijspeert, "Movement generation using dynamical systems: a humanoid robot performing a drumming task," in *Proceedings of the IEEE-RAS International Conference on Humanoid Robots (HUMANOIDS06)*, 2006.

[5] S.-M. Khansari-Zadeh and A. Billard, "Imitation learning of globally stable non-linear point-to-point robot motions using nonlinear programming," in *Proceeding of theIEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2010.2010, 2010.

[6] E. Gribovskaya, S. M. Khansari-Zadeh, and A. Billard, "Learning nonlinear multivariate dynamics of motion in robotic manipulators," *The International Journal of Robotics Research*, pp. 1–37, 2010.

[7] S.-M. Khansari-Zadeh and A. Billard, "BM: An iterative algorithm to learn stable non-linear dynamical systems with gaussian mixture models," *in* Proceeding of the International Conference on Robotics and Automation (ICRA) *pp*, pp. 2381–2388, 2010.

[8] G. McLachlan and D. Peel, *Finite Mixture Models.* : Wiley, 2000.

[9] D. Cohn and Z. Ghahramani, "Active learning with statistical models," *Artificial Intelligence Research*, pp. 129–145, 1996.

[10] A. Dempster and N. L. D. Rubin, "Maximum likelihood from incomplete data via the em algorithm," *Journal of the Royal Statistical Society B*, vol. 39, no. 1, pp. 1–38, 1977.

[11] S. Vijayakumar and S. Schaal, "Locally weighted projection regression: An o(n) algorithm for incremental real time learning in high dimensional space," in *Proceedings of the Seventeenth International Conference on Machine Learning (ICML)*, 2000.

[12] C. Rasmussen and C. Williams, *Gaussian processes for machine learning.* : Springer, 2006.

[13] S. Calinon, F. Guenter, and A. Billard, "On learning, representing and generalizing a task in a humanoid robot," *IEEE transactions on systems, man and cybernetics*, vol. 37, no. 2, pp. 286–298, 2007.

[14] M. S. Bazaraa, H. Sherali, and C. Shetty, "Nonlinear programming: Theory and algorithms," in *3rd Edition*, Ed. John & SonsEd. John Wiley & Sons: Wiley, 2006.

[15] R. D. Robinett, *Applied dynamic programming for optimization of dynamical systems*. SIAM, 2005.

[16] C. T. Kelley, "Line search methods and the armijo rule," *in* Iterative Methods for Optimization, vol. 3, no. 2, pp. 40–52, 1999.

[17] G. Schwarz, "Estimating the dimension of a model," *Annals of Statistics*, pp. 461–464, 1978.

[18] S. Kim, E. Gribovskaya, and A. Billard, "Learning Motion Dynamics to Catch a Moving Object," in *the 10th IEEE-RAS International Conference on Humanoid Robots*, 2010.

[19] H. Jaeger, "The echo state approach to analysing and training recurrent neural networks," *submitted for publication*, 2001.

[20] W. Maass, T. Natschlager, and H. Markram, "Real-time computing without stable states: A new framework for neural computation based on perturbations," *Neural computation*, vol. 14, no. 11, pp. 2531–2560, 2002.

[21] B. Schrauwen, D. Verstraeten, and J. Van Campenhout, "An overview of reservoir computing: theory, applications and implementations," in *Proceedings of the 15th European Symposium on Artificial Neural Networks*, pp. 471–482, Citeseer, 2007.

[22] D. Sussillo and L. Abbott, "Generating coherent patterns of activity from chaotic neural networks," *Neuron*, vol. 63, no. 4, pp. 544–557, 2009.

[23] J. J. Steil, R. F. Reinhart, and M. Rolf, "Neural dynamic movement primitives based on associative reservoir learning," *submitted to Adaptive Behavior*, 2010.

[24] D. Wolpert and M. Kawato, "Multiple paired forward and inverse models for motor control," *Neural Networks*, pp. 1317–1329, 1998.

[25] J. J. Steil, "Backpropagation-decorrelation: recurrent learning with O(N) complexity," in *Proc. IJCNN*, vol. 1, pp. 843–848, 2004.

[26] J. Triesch, "A gradient rule for the plasticity of a neuron's intrinsic excitability," in *Proc. ICANN*, pp. 65–79, 2005.

[27] S. Wrede, A. Nordmann, A. Lemme, S. Rüther, M. Johannfunke, A. Weirich, and J. J. Steil, "A flexible and intelligent robot control framework," in *20. GMA-FA Workshop on Computational Intelligence*, 2010. submitted.

[28] M. Rolf, J. J. Steil, and M. Gienger, "Efficient exploration and learning of whole body kinematics," in *IEEE 8th International Conference on Development and Learning*, 2009.

[29] K. Neumann, M. Rolf, J. J. Steil, and M. Gienger, "Learning inverse kinematics for pose-constraint bi-manual movements," in *Simulation of Adaptive Behavior – SAB*, 2010. in press.

[30] M. Rolf, J. J. Steil, and M. Gienger, "Learning flexible full body kinematics for humanoid tool use," in *ECSIS Symp. on LAB-RS*, 2010. in press.

[31] J. J. Steil, "Online reservoir adaptation by intrinsic plasticity for backpropagation-decorrelation and echo state learning," *Neural Networks*, vol. 20, no. 3, pp. 353–364, 2007.

[32] R. F. Reinhart and J. J. Steil, "Attractor-based computation with reservoirs for online learning of inverse kinematics," in *Proc. ESANN*, pp. 257–262, 2009.

[33] J. J. Steil, "Online stability of backpropagation-decorrelation recurrent learning," *Neurocomputing*, vol. 69, pp. 642–650, Mar 2006.

[34] M. Lukosevicius and H. Jaeger, "Reservoir computing approaches to recurrent neural network training," *Computer Science Review*, pp. 127–149, 2009.

[35] M. Toussaint, "Robot trajectory optimization using approximate inference," in *ICML '09: Proceedings of the 26th Annual International Conference on Machine Learning*, (New York, NY, USA), pp. 1049–1056, ACM, 2009.

[36] E. Todorov and W. Li, "A generalized iterative lqg method for locally-optimal feedback control of constrained nonlinear stochastic systems," 2005.

[37] E. Theodorou, Y. Tassa, and E. Todorov, "Stochastic differential dynamic programming," 2010.

[38] D. Mitrovic, S. Klanke, and S. Vijayakumar, "Optimal control with adaptive internal dynamics models," 2008.

[39] D. Mitrovic, S. Klanke, and S. Vijayakumar, "Adaptive optimal feedback control with learned internal dynamics models," 2010.

[40] M. Toussaint and K. Rawlik, "Approximate inference control, submitted for publication," 2010.

[41] M. Hoffman, H. Kueck, N. de Freitas, and A. Doucet, "New inference strategies for solving markov decision processes using reversible jump mcmc," in *Proceedings of the Proceedings of the Twenty-Fifth Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-09)*, (Corvallis, Oregon), pp. 223–231, AUAI Press, 2009.

[42] W. Li and E. Todorov, "Iterative linear quadratic regulator design for nonlinear biological movement systems," 2004.

[43] C. Atkeson and B. Stephens, "Multiple balance strategies from one optimization criterion," 2007.

[44] V. Heidrich-Meisner and C. Igel, "Neuroevolution strategies for episodic reinforcement learning," *J. Algorithms*, vol. 64, no. 4, pp. 152–168, 2009.

[45] D. Nguyen-tuong, J. Peters, M. Seeger, and B. Schölkopf, "Learning inverse dynamics: a comparison."

[46] A. J. Ijspeert, J. Nakanishi, and S. Schaal, "Learning attractor landscapes for learning motor primitives," in *in Advances in Neural Information Processing Systems*, pp. 1523–1530, MIT Press, 2003.

[47] L. Righetti, J. Buchli, and A. J. Ijspeert, "Adaptive Frequency Oscillators and Applications," *The Open Cybernetics and Systemics Journal*, vol. 3, pp. 64–69, 2009.

[48] J. Buchli and A. J. Ijspeert, "Self-organized adaptive legged locomotion in a compliant quadruped robot," *Autonomous Robots*, vol. 25, no. 4, pp. 331–347, 2008.

[49] L. Righetti, J. Buchli, and A. J. Ijspeert, "Dynamic hebbian learning in adaptive frequency oscillators," *Physica D*, vol. 216, no. 2, pp. 269–281, 2006.

[50] L. Righetti, J. Buchli, and A. J. Ijspeert, "Dynamic fourier series decomposition with pools of strongly coupled adaptive frequency oscillators." Unpublished.

[51] D. Sussillo and L. Abbott, "Generating coherent patterns of activity from chaotic neural networks," *Neuron*, vol. 63, pp. 544–557, 2009.

[52] H. Jaeger, "The "echo state" approach to analysing and training recurrent neural networks," Tech. Rep. GMD Report 148, German National Research Center for Information Technology, 2001.

[53] W. Maass, T. Natschläger, and H. Markram, "Real-time computing without stable states: A new framework for neural computation based on perturbations," *Neural Computation*, vol. 14, no. 11, pp. 2531–2560, 2002.

[54] D. Verstraeten, B. Schrauwen, M. D'Haene, and D. Stroobandt, "An experimental unification of reservoir computing methods," *Neural Networks*, vol. 20, no. 3, pp. 391–403, 2007.

[55] A. Gams, A. Ijspeert, S. Schaal, and J. Lenarcic, "On-line learning and modulation of periodic movements with nonlinear dynamical systems," *Autonomous Robots*, vol. 27, no. 1, pp. 3–23, 2009.